## Automated vs manual testing

Automated Testing refers to the use of software tools to execute pre-scripted tests on a software application without human intervention. It involves creating test scripts that validate the functionality, performance, and reliability of the software by comparing actual outcomes with expected results.

Manual Testing, on the other hand, is a process where human testers interact directly with the software to evaluate its quality. Testers manually execute test cases without the help of automation tools, relying on their judgment to identify bugs and assess usability.

## Principles of Automated Testing

1. Efficiency and Speed: Automated testing is designed to execute repetitive tests quickly and consistently.
2. Scalability: It allows for running multiple tests in parallel, making it highly scalable.
3. Consistency: Automated tests perform the same steps precisely every time, reducing human error.
4. Improved Quality: By enabling frequent and comprehensive testing, automation helps improve overall software quality.
5. Risk Reduction: Automated tests can catch regressions and errors before they reach production, reducing risk

Automated testing is best suited for:

1. Repetitive Tasks: Tests that need to be run frequently over time.
2. Regression Testing: When frequent regression testing is required.
3. Large Codebases: Software with complex functionalities or large codebases.

## Principles of Manual Testing

1. Human Insight: Manual testing involves direct human observation and judgment.
2. Flexibility: It allows for exploratory testing and adapting to unexpected behaviors.
3. Visual Feedback: Manual testers can assess the user interface and overall user experience.
4. Hands-on Understanding: It provides deeper insights into real-time system behavior and usability

Manual testing is more appropriate for:

1. Exploratory Testing: When new functionalities need to be explored or understood.
2. Usability Testing: Assessing how user-friendly an application is.
3. Short-Term Projects: When the project doesn't justify the cost of automation.

**References**

<https://www.qaratest.com/blogs/key-principles-of-automation-testing-a-guide-to-efficient-testing-with-qara-enterprise>

<https://semaphore.io/blog/test-automation>

<https://www.browserstack.com/guide/manual-testing-tutorial>

<https://www.globalapptesting.com/blog/automated-testing-vs-manual-testing>

<https://www.leapwork.com/blog/manual-vs-automated-testing>

## **User Stories: Select Car**

## **As a** user,

## **I want to** be able to select a car from a dropdown list,

## **So that** I can view details about the car such as MPG, road tax, and insurance group.

## **Acceptance Criteria:**

## The dropdown menu should show a list of available cars.

## Upon selecting a car, the car's details (MPG, tax, insurance group) should be displayed.

## The car's image should be shown upon selection.

## **2. User Story: Input Age**

## **As a** user,

## **I want to** enter my age in the form,

## **So that** the application can provide an accurate insurance estimate based on my age.

## **Acceptance Criteria:**

## The input field should only allow numeric values between 18 and 100.

## If an invalid age is entered (less than 18 or more than 100), the user should see an alert message.

## **3. User Story: Input Annual Mileage**

## **As a** user,

## **I want to** input my annual mileage,

## **So that** the application can estimate my fuel cost based on how many miles I drive each year.

## **Acceptance Criteria:**

## The mileage input field should only accept numeric values and a minimum of 500 miles.

## If the entered mileage is less than 500, the user should see an alert message.

## **4. User Story: Submit Form and Calculate Costs**

## **As a** user,

## **I want to** submit the form with my age, mileage, and car selection,

## **So that** I can get an estimate of my car's annual running costs, including fuel, road tax, and insurance.

## **Acceptance Criteria:**

## The form should submit and prevent the page from reloading.

## The application should display the estimated annual fuel cost, road tax, and insurance cost.

## The total running cost should be calculated and displayed clearly.

## **5. User Story: Display Error Messages**

## **As a** user,

## **I want to** see error messages if I have not selected a car, entered an invalid age, or entered an invalid mileage,

## **So that** I can correct the inputs and try again.

## **Acceptance Criteria:**

## If any required field is missing or invalid (age, mileage, car selection), the user should see an alert with a clear message indicating the error.

## **6. User Story: Responsive Design**

## **As a** user,

## **I want to** access the application on any device (desktop, tablet, mobile),

## **So that** I can use the car cost estimator wherever I am.

## **Acceptance Criteria:**

## The application should be responsive and adapt to different screen sizes.

## The layout should adjust based on the device screen width, maintaining usability and accessibility.

## **7. User Story: Accessibility**

## **As a** user with a visual impairment,

## **I want to** be able to use the car running cost estimator with screen readers,

## **So that** I can interact with the application and receive the necessary information.

## **Acceptance Criteria:**

## All important elements should be properly labeled with aria attributes (e.g., aria-labelledby, aria-describedby).

## Form fields should be properly labeled and described for screen readers.

## Visually hidden text should be available to screen readers to improve accessibility.

## **8. User Story: View Car Image**

## **As a** user,

## **I want to** see an image of the car I select,

## **So that** I can visually confirm the car I'm choosing.

## **Acceptance Criteria:**

## When a car is selected from the dropdown, its image should appear in the car details section.

## The image should be responsive and scale according to the size of the page.

## **9. User Story: Reset the Form**

## **As a** user,

## **I want to** reset the form and start over,

## **So that** I can quickly calculate the running costs for a different car or input new data.

## **Acceptance Criteria:**

## A "reset" button should be available that clears all input fields, car selection, and result display.

## The user should be able to re-enter new values without issues.

## **10. User Story: Clear Results**

## **As a** user,

## **I want to** be able to clear the results section after viewing my running costs,

## **So that** I can start a new calculation without the previous data interfering.

## **Acceptance Criteria:**

## The results section should be able to be cleared after each calculation.

## The results should be hidden until a new calculation is made.

**1. User Story: Select Car**

* **As a** user,
* **I want to** be able to select a car from a dropdown list,
* **So that** I can view details about the car such as MPG, road tax, and insurance group.

## 

## 

**Acceptance Criteria:**

* The dropdown menu should show a list of available cars.
* Upon selecting a car, the car's details (MPG, tax, insurance group) should be displayed.
* The car's image should be shown upon selection.

**2. User Story: Input Age**

* **As a** user,
* **I want to** enter my age in the form,
* **So that** the application can provide an accurate insurance estimate based on my age.

**Acceptance Criteria:**

* The input field should only allow numeric values between 18 and 100.
* If an invalid age is entered (less than 18 or more than 100), the user should see an alert message.

## 

**3. User Story: Input Annual Mileage**

* **As a** user,
* **I want to** input my annual mileage,
* **So that** the application can estimate my fuel cost based on how many miles I drive each year.

**Acceptance Criteria:**

* The mileage input field should only accept numeric values and a minimum of 500 miles.
* If the entered mileage is less than 500, the user should see an alert message.

## 

**4. User Story: Submit Form and Calculate Costs**

* **As a** user,
* **I want to** submit the form with my age, mileage, and car selection,
* **So that** I can get an estimate of my car's annual running costs, including fuel, road tax, and insurance.

**Acceptance Criteria:**

* The form should submit and prevent the page from reloading.
* The application should display the estimated annual fuel cost, road tax, and insurance cost.
* The total running cost should be calculated and displayed clearly.

## 

**5. User Story: Display Error Messages**

* **As a** user,
* **I want to** see error messages if I have not selected a car, entered an invalid age, or entered an invalid mileage,
* **So that** I can correct the inputs and try again.

**Acceptance Criteria:**

* If any required field is missing or invalid (age, mileage, car selection), the user should see an alert with a clear message indicating the error.

**6. User Story: Responsive Design**

* **As a** user,
* **I want to** access the application on any device (desktop, tablet, mobile),
* **So that** I can use the car cost estimator wherever I am.

**Acceptance Criteria:**

* The application should be responsive and adapt to different screen sizes.
* The layout should adjust based on the device screen width, maintaining usability and accessibility.

**Mobile view**

## 

**Desktop view**

## 

**7. User Story: Accessibility**

* **As a** user with a visual impairment,
* **I want to** be able to use the car running cost estimator with screen readers,
* **So that** I can interact with the application and receive the necessary information.

**Acceptance Criteria:**

* All important elements should be properly labeled with aria attributes (e.g., aria-labelledby, aria-describedby).
* Form fields should be properly labeled and described for screen readers.
* Visually hidden text should be available to screen readers to improve accessibility.

## 

**8. User Story: View Car Image**

* **As a** user,
* **I want to** see an image of the car I select,
* **So that** I can visually confirm the car I'm choosing.

**Acceptance Criteria:**

* When a car is selected from the dropdown, its image should appear in the car details section.
* The image should be responsive and scale according to the size of the page.

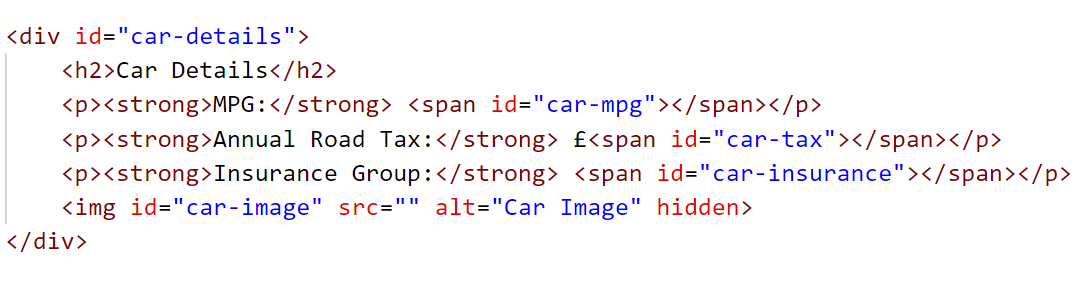
## **Testing plan: Select Car**

## Test plan

| **Test Number** | **Type of Test** | **Test Description** | **Expected Result** | **Actual Result** |
| --- | --- | --- | --- | --- |
| 1 | Functional Test | Verify car selection dropdown displays the list of available cars | A dropdown list of cars is displayed, with the correct car names in the list. |  |
| 2 | Functional Test | Verify car details display upon car selection | Car details (MPG, tax, insurance group) are displayed, and the car image appears. |  |
| 3 | Validation Test | Verify that age input only accepts numbers between 18 and 100 | The age input field accepts values between 18 and 100 and displays an error if the value is outside this range. |  |
| 4 | Validation Test | Verify that annual mileage input only accepts numbers greater than or equal to 500 | The mileage input accepts values of 500 and above and shows an error if below 500. |  |
| 5 | Functional Test | Verify error alert when age input is empty or invalid | If the age input is empty or invalid (less than 18 or greater than 100), an alert should be displayed asking for a valid age. |  |
| 6 | Functional Test | Verify error alert when mileage input is empty or below 500 | If the mileage input is empty or less than 500, an alert should appear requesting a valid mileage. |  |
| 7 | Functional Test | Verify error alert when no car is selected | If no car is selected from the dropdown, an alert should prompt the user to select a car. |  |
| 8 | Functional Test | Verify that the total cost is calculated correctly | The total cost is calculated by summing fuel cost, tax, and insurance, and it should match the expected value based on inputs. |  |
| 10 | UI Test | Verify responsive design on mobile | The form adjusts to fit small screen sizes like mobile devices (width ≤ 600px). |  |
| 11 | UI Test | Verify form displays correctly on large screen (desktop) | The form width should adjust properly on large screens (e.g., 1000px wide). |  |
| 12 | Accessibility Test | Verify form fields are properly labeled for screen readers | All form fields have aria-label, aria-describedby, and are correctly labeled for screen readers. | Via Chrome Developer tools Source for testing this <https://stackoverflow.com/questions/56427619/how-to-test-your-aria-tags-on-your-website> |
| 13 | Functional Test | Verify the car image is displayed correctly after car selection | The car image should appear and scale correctly with the page size. |  |
| 14 | Usability Test | Verify that the "Calculate Costs" button works after filling in the form | When the form is filled in correctly, clicking "Calculate Costs" should trigger the cost calculation and display results. |  |
| 16 | Edge Case Test | Verify that the application handles a high input mileage value | The application should still calculate the costs correctly even with a high mileage value. |  |
| 17 | Edge Case Test | Verify that the application handles a low input mileage value (e.g., 500) | The application should calculate the costs for a low mileage value, ensuring no issues arise. |  |
| 18 | Functional Test | Verify the calculation of fuel cost | The fuel cost should be calculated as (annual mileage / mpg) \* 1.50 and display a value based on correct calculations. |  |
| 19 | Functional Test | Verify the calculation of insurance cost | The insurance cost is calculated based on the insurance group, age, and a random factor. |  |
| 20 | Functional Test | Verify the calculation of road tax | The road tax value should be displayed exactly as it is set in the car data (e.g., £150 for Ford Fiesta). |  |
| 21 | Functional Test | Verify that the result section is updated only after the form is submitted | The result section should only be populated after the form is submitted and not before. |  |

## Problems encountered during the development process:

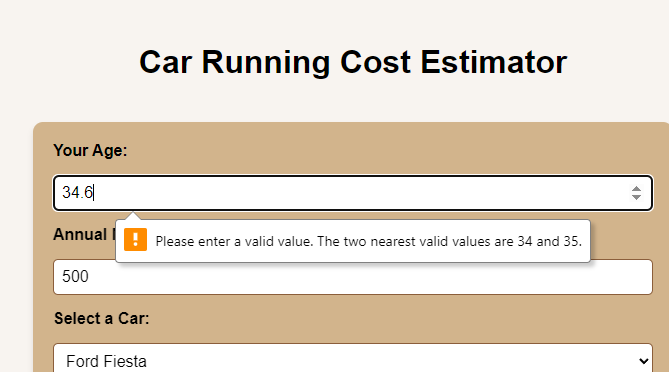
After developing the Javascript page I couldn’t load the car details on the webpage. This proved to be because I hadn’t established the names for the HTML tags correctly.



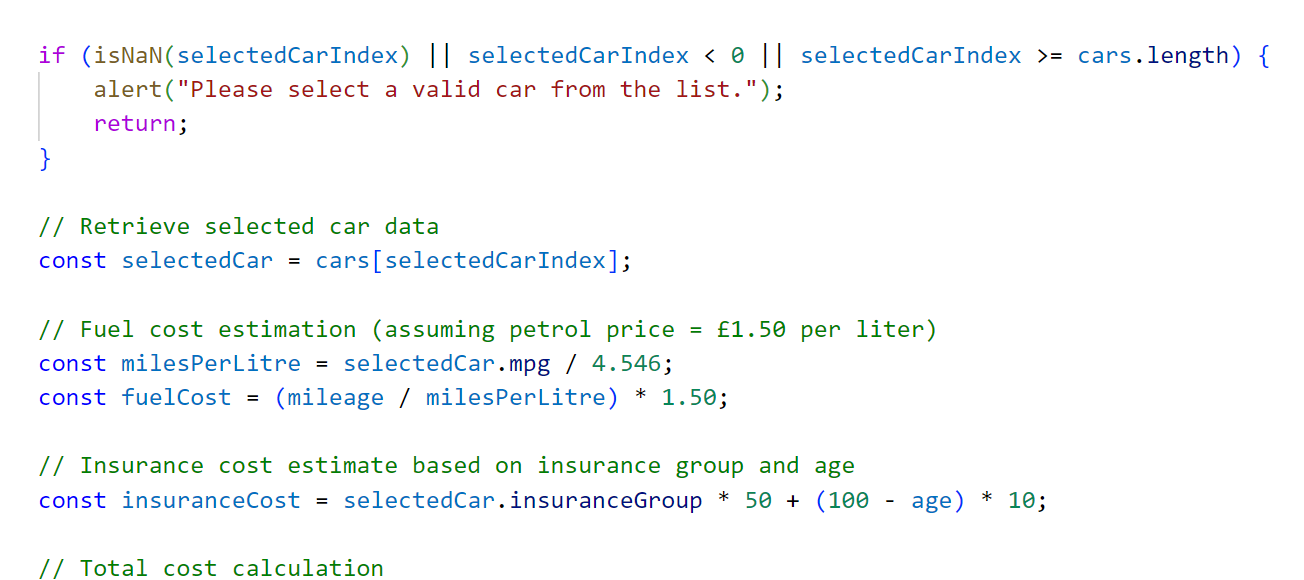
After modifying the id tags this was fine.

I noticed that I could crash the program by entering floating point numbers as the code uses parseInt, which is the most likely input.

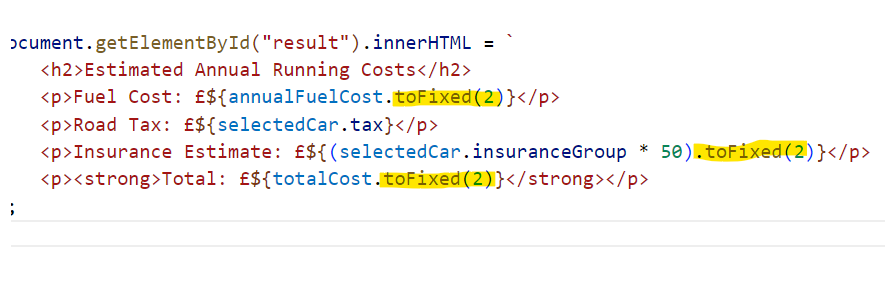
const selectedIndex = parseInt(carSelect.value, 10);



I changed the order of the validation checks to proceed the calculation, so it would only attempt a sum if all values were valid



My initial calculations weren’t rounding to 2 decimal places. I eventually found I was trying to round in Python rather than Javascript!



## 

## 

## **Validation Testing**

### Javascript report

## <https://www.site24x7.com/tools/javascript-validator.html>

## 

## Line 2, Column 1: 'const' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

## Line 19, Column 1: 'const' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

## Line 20, Column 25: 'arrow function syntax (=>)' is only available in ES6 (use 'esversion: 6').

## Line 21, Column 5: 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

## Line 28, Column 39: 'arrow function syntax (=>)' is only available in ES6 (use 'esversion: 6').

## Line 29, Column 5: 'const' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

## Line 37, Column 9: 'const' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

## Line 51, Column 5: 'const' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

## Line 52, Column 5: 'const' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

## Line 53, Column 5: 'const' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

## Line 72, Column 5: 'const' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

## Line 75, Column 5: 'const' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

## Line 76, Column 5: 'const' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

## Line 79, Column 5: 'const' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

## Line 82, Column 5: 'const' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

## Line 85, Column 51: 'template literal syntax' is only available in ES6 (use 'esversion: 6')

## <https://validator.w3.org/nu/#file>

## 

## Thankfully the fix for this was only one line of code at the start if the script to tell jshint to use version ES6

## /\* jshint esversion: 6 \*/

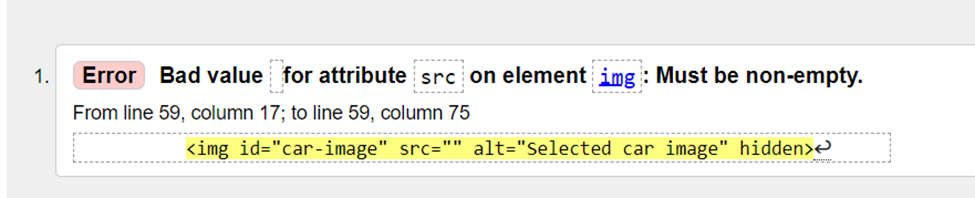
## 

## Source for this fix

<https://stackoverflow.com/questions/27441803/why-does-jshint-throw-a-warning-if-i-am-using-const>

### HTML

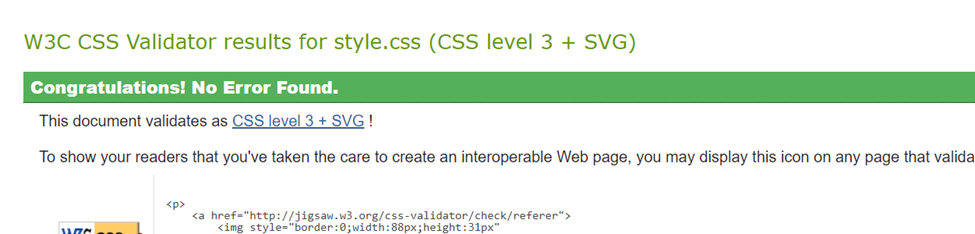
<https://validator.w3.org/nu/#file>



To pass the validation I changed the code so that the image displayed in a block tag.

const carImage = document.getElementById("car-image"); carImage.src = selectedCar.img; // Make the image visible by setting display to block carImage.style.display = "block"; // Show the car details box document.getElementById("car-details").style.display = "block";

CSS



<https://jigsaw.w3.org/css-validator/validator>