CERTIFICATE IV WEB BASED TECHNOLOGY - FRONT END DEVELOPMENT -SCRIPTING

DEVELOPMENT LOG - FLOW CALCULATOR APPLICATION

BY OSAMU MOROZUMI

Development

Background and inspiration

The idea for the application came from my work in the horticulture industry. I developed a number of spreadsheets to help calculate flow for irrigation design. Although these spreadsheets worked fine in the office, they were unusable in the field. I thought it may be worth transferring the idea to a web application which would be optimised for mobile usage.

I decided to develop a prototype first, which would demonstrate how the user interface would work, but limit the data to one sprinkler type: the Rainbird 3500 series gear rotator.

Planning and development

Layout

I based the design on the spreadsheets and used a table layout to display the flow and nozzle data. I deliberately kept the user interface as simple as possible so I could focus on developing the JavaScript component.

Data storage and retrieval

As there is a reasonably large amount of data to work with in this application I needed a strict structure to store it uniformly. I chose to use a large object literal, much like a JSON file. This meant I could retrieve specific chunks of data using keys, but could still iterate through the object using the Object.keys and Object.values methods.

File structure

I tried to organise the code into files which shared common characteristics such as data storage, utility functions, error checking and DOM manipulation. This proved difficult as the utility of different functions often overlapped categories.

Function composition

I endeavoured to write pure, reusable functions, but this proved very difficult when dealing with the DOM, resulting in a lot of procedural code.

Testing

The most important aspects to test were:

1. Setting the flow rate and pressure retrieved the correct data and reflected in the flow data table
2. Adjusting the number of nozzles increased the total flow for that nozzle and added the correct amount to the total flow
3. Adjusting the number of nozzles updated the number of stations accurately
4. Changing the pressure and available flow updated the total flow and necessary stations accurately

Other important aspects to test:

1. UI doesn't display any unusual values such as negative values or javascript values (such as NaN)
2. Error functions fire at correct times and not unexpectedly
3. Social media sharing links to correct platform without errors

Testing for accurate calculations

To test that the flow data was accurate I needed to compare the results which my application produced to manual calculations based on the Rainbird technical specifications brochure.

For example:

An irrigation area with 2.0 bar pressure and 30 litres a minute available flow with:

- 3 x 0.75 type nozzles
- 4 x 1.0 type nozzles
- 1 x 1.5 type nozzle

would require a total of 23.63 litres a minute of flow and could run one 1 station.

If the pressure were reduced to 1.7 bar the same area would require a total of 21.77 and would still run on 1 station.

If the flow were reduced to 20 litres a minute then at either 1.7 bar or 2.0, it would require 2 stations to run.

Testing the application with this data produces the same results as manual calculation, only a lot faster!

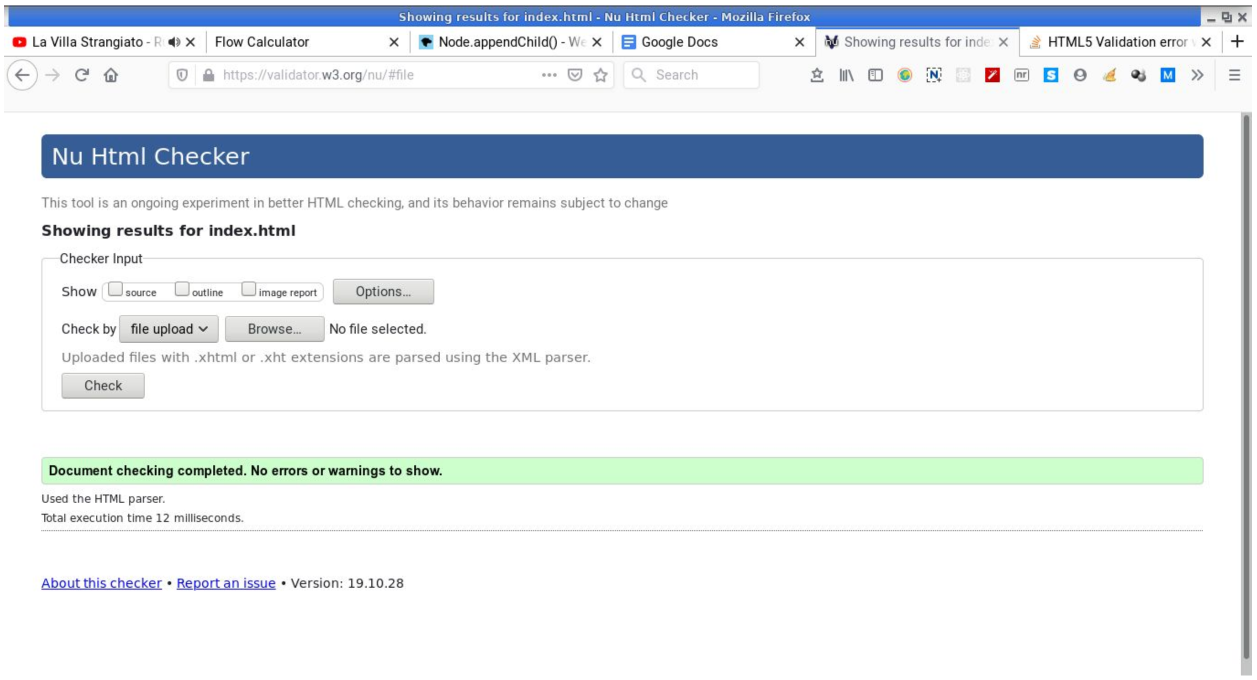
Cross browser testing

All important aspects of the application were tested in Google Chrome and Mozilla Firefox. Each aspect tested satisfactorily without errors.

Video evidence in file *cross-browser-testing.mp4*

Validating markup

Markup tested using online service https://validator.w3.org/. Screenshot below:


Nu Html Checker

This tool is an ongoing experiment in better HTML checking, and its behavior remains subject to change

Showing results for index.html

Checker Input

Show ☐ source ☐ outline ☐ image report

Check by file upload · No file selected.

Uploaded files with .xhtml or .xht extensions are parsed using the XML parser.

Document checking completed. No errors or warnings to show.

Used the HTML parser.

Total execution time 12 milliseconds.

About this checker • Report an issue • Version: 19.10.28

Showing results for index.html - Nu Html Checker - Mozilla Firefox
https://validator.w3.org/nu/#file
Options...
Browse...
Check

Debugging

Debugging log is organised by file name. Larger screenshots in *documents/debug-screenshots/*

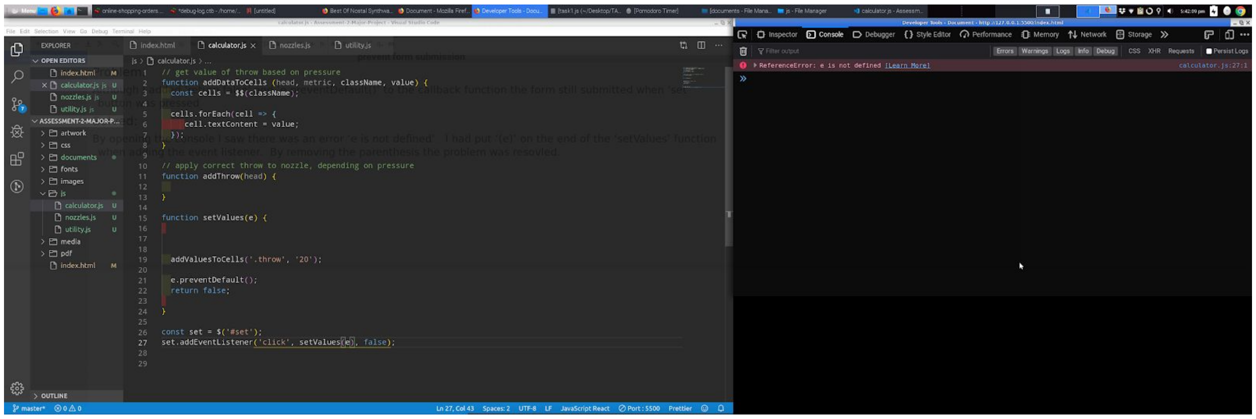
Init.js

Problem

Although I added ‘return false' and ‘e.preventDefault()’ to the callback function the form still submitted when ‘set’ button was pressed

Solved

By opening the console I saw there was an error ‘e is not defined’. I had put ‘(e)’ on the end of the ‘setValues’ function when adding the event listener. By removing the parentheses the problem was resolved.



```
// get value of throw based on pressure
function addDataToCells (head, metric, className, value) {
  const cells = $$(className);

  cells.forEach(cell => {
    cell.textContent = value;
  });
}

// apply correct throw to nozzle, depending on pressure
function addThrow(head) {

}

function setValues(e) {



  addValuesToCells('.throw', '20');

  e.preventDefault();
  return false;

}

const set = $('#set');
set.addEventListener('click', setValues(e), false);
```

```
ReferenceError: e is not defined [Learn More]        calculator.js:27:1
```

Problem

Currently the 'adjustSingleFlow' function can not be used in isolotion from an event listener. I want to be able to call the function and recalculate the total flow of each nozzle whenever the pressure or available flow is changed.

Solution

The function needs to be refactored so it can be called separately from any event. A separate function must be created for the event listener which doesn't have any parameters apart from the event itself. I did this with all event listener call backs and separated them into the 'init.js' file.

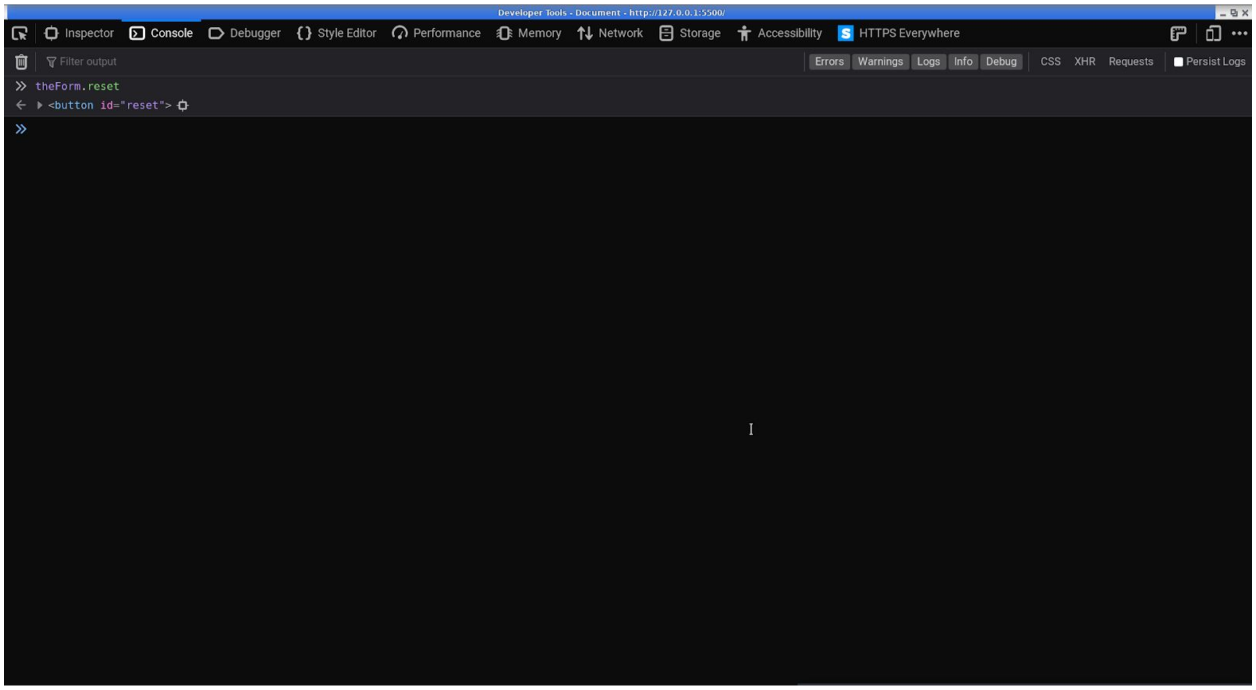
No screenshot available see both init.js and calculator.js

Problem

Universal reset button not working and doesn't prevent form submission

Solved

Using the console to inspect the ‘reset’ property of the form I was able to see that it was set to the reset button which had the id of ‘reset’. By changing the id attribute of the reset button I was able to manually call the form's reset method, however it failed when attached called by the ‘click’ event. I tried using a debugger statement but the form submitted before opening the debugger. I then experimented using different event listeners until I tried the ‘reset’ event which resolved the issue.



```
>> theForm.reset
<- ▸ <button id="reset">
>>
```

Problem

When resetting the page the inputs were not being set to disabled.

Solved

By setting all nozzle adjuster inputs to 'disabled' in the markup and calling the enable inputs function as the page loaded. I also needed to add conditional logic to the 'preventExecution' function to avoid 'e is not defined' errors.

No screenshot available. See files init.js and index.html

Problem

Nozzle amount values lingered after page reloads

Solved

Add 'resetAllValues' to window.onload event

No screenshot available. See line 64.

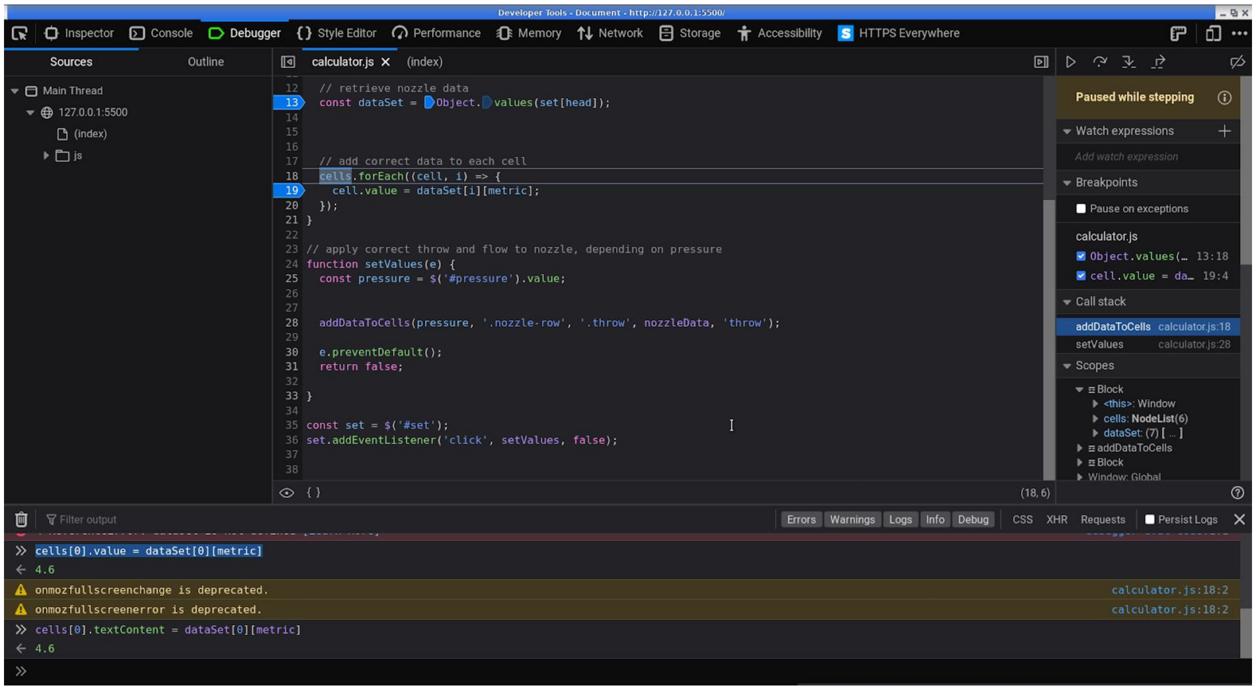
calculator.js

Problem

Function for retrieving correct nozzle data and applying it to the table cells was failing.

Solved

Using debugger I paused the ‘setValues’ function after defining the ‘pressure’ variable. I then typed ‘pressure’ into the console to see what it would return. It returned the ‘select’ input as a DOM node rather than a string value. I changed the ‘pressure’ variable definition to the value of the select input, rather than the select input itself.



```
  // retrieve nozzle data
  const dataSet = Object.values(set[head]);



  // add correct data to each cell
  cells.forEach((cell, i) => {
    cell.value = dataSet[i][metric];
  });
}

// apply correct throw and flow to nozzle, depending on pressure
function setValues(e) {
  const pressure = $('#pressure').value;


  addDataToCells(pressure, '.nozzle-row', '.throw', nozzleData, 'throw');

  e.preventDefault();
  return false;

}

const set = $('#set');
set.addEventListener('click', setValues, false);
```

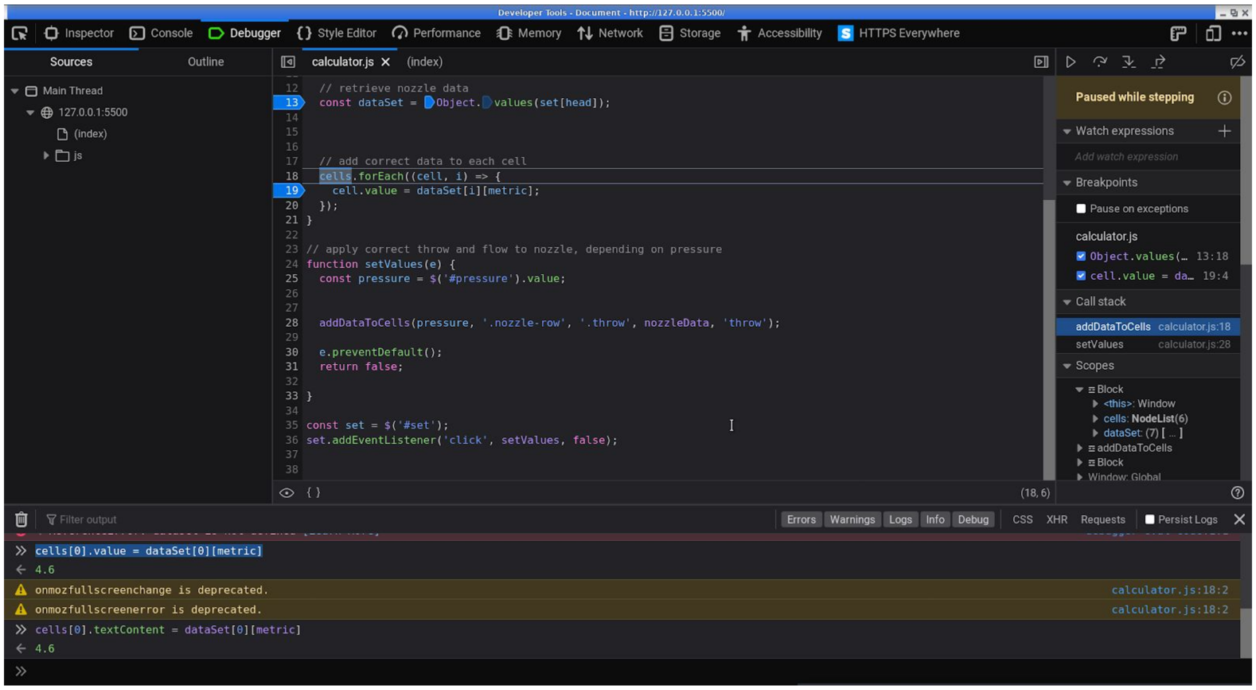
```
>> cells[0].value = dataSet[0][metric]
← 4.6
⚠ onmozfullscreenchange is deprecated.                    calculator.js:18:2
⚠ onmozfullscreenerror is deprecated.                     calculator.js:18:2
>> cells[0].textContent = dataSet[0][metric]
← 4.6
>>
```

Problem

Correct data was not getting inserted into each cell. I was using a forEach loop to loop through the cells.

Solved

Again using the debugger I paused the program in the middle of the function so I could access all the variables. I tried executing the action which the loop was performing manually to see what was happening ('cells[0].value = dataSet[0][metric]'). This returned the value I was expecting: ‘4.6’ but it still wasn't rendering on the screen. I realised I should have been using 'cells[i].textContent' instead of ‘value’. I changed the program accordingly and the data rendered as expected.



```
  // retrieve nozzle data
  const dataSet = Object.values(set[head]);



  // add correct data to each cell
  cells.forEach((cell, i) => {
    cell.value = dataSet[i][metric];
  });
}

// apply correct throw and flow to nozzle, depending on pressure
function setValues(e) {
  const pressure = $('#pressure').value;


  addDataToCells(pressure, '.nozzle-row', '.throw', nozzleData, 'throw');

  e.preventDefault();
  return false;

}

const set = $('#set');
set.addEventListener('click', setValues, false);
```

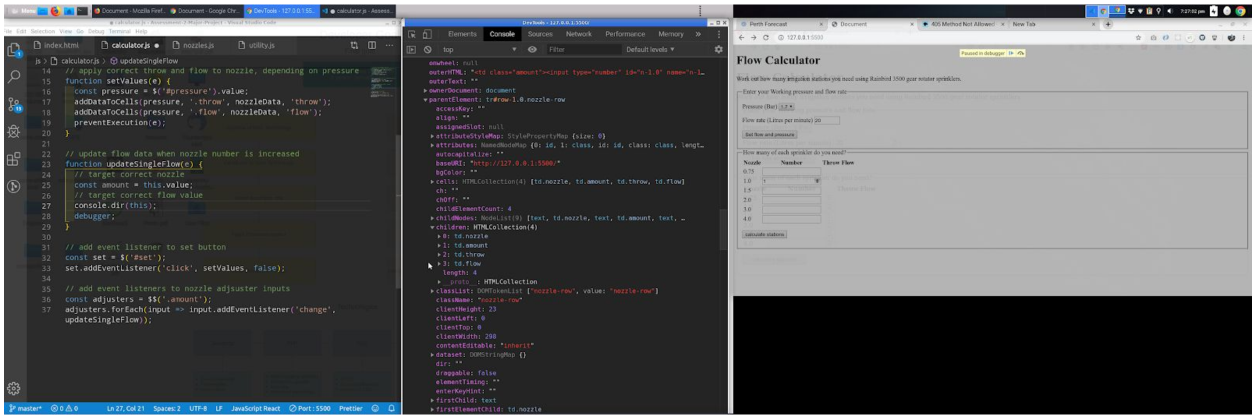
```
>> cells[0].value = dataSet[0][metric]
← 4.6
⚠ onmozfullscreenchange is deprecated.                         calculator.js:18:2
⚠ onmozfullscreenerror is deprecated.                          calculator.js:18:2
>> cells[0].textContent = dataSet[0][metric]
← 4.6
>>
```

Problem

While applying an event listener function to the number inputs (.amount) I wasn't sure how to then target the flow data cell from the same scope.

Solved

Pausing function with debugger I inspected the element's attributes until I found the ‘parentElement’ attribute. I opened this up and found ‘children’ attribute. The element I wanted to target was found in this list.



```
14  // apply correct throw and flow to nozzle, depending on pressure
15  function setValues(e) {
16    const pressure = $('#pressure').value;
17    addDataToCells(pressure, '.throw', nozzleData, 'throw');
18    addDataToCells(pressure, '.flow', nozzleData, 'flow');
19    preventExecution(e);
20  }
21
22  // update flow data when nozzle number is increased
23  function updateSingleFlow(e) {
24    // target correct nozzle
25    const amount = this.value;
26    // target correct flow value
27    console.dir(this);
28    debugger;
29  }
30
31  // add event listener to set button
32  const set = $('#set');
33  set.addEventListener('click', setValues, false);
34
35  // add event listeners to nozzle adjsuster inputs
36  const adjusters = $$('.amount');
37  adjusters.forEach(input => input.addEventListener('change',
    updateSingleFlow));
```

```
  onwheel: null
  outerHTML: "<td class="amount"><input type="number" id="n-1.0" name="n-1…
  outerText: ""
▸ ownerDocument: document
▾ parentElement: tr#row-1.0.nozzle-row
    accessKey: ""
    align: ""
    assignedSlot: null
  ▸ attributeStyleMap: StylePropertyMap {size: 0}
  ▸ attributes: NamedNodeMap {0: id, 1: class, id: id, class: class, lengt…
    autocapitalize: ""
    baseURI: "http://127.0.0.1:5500/"
    bgColor: ""
  ▸ cells: HTMLCollection(4) [td.nozzle, td.amount, td.throw, td.flow]
    ch: ""
    chOff: ""
    childElementCount: 4
  ▸ childNodes: NodeList(9) [text, td.nozzle, text, td.amount, text, …
  ▾ children: HTMLCollection(4)
    ▸ 0: td.nozzle
    ▸ 1: td.amount
    ▸ 2: td.throw
    ▸ 3: td.flow
      length: 4
    ▸ __proto__: HTMLCollection
  ▸ classList: DOMTokenList ["nozzle-row", value: "nozzle-row"]
    className: "nozzle-row"
    clientHeight: 23
    clientLeft: 0
    clientTop: 0
    clientWidth: 298
    contentEditable: "inherit"
  ▸ dataset: DOMStringMap {}
    dir: ""
    draggable: false
    elementTiming: ""
    enterKeyHint: ""
  ▸ firstChild: text
  ▸ firstElementChild: td.nozzle
```

Problem

As I was increasing the number of nozzles, the value which the total flow referred to was also increased

Solved

I needed to store the current nozzle data separately from the data in the table to avoid mutating it. This meant creating a container to keep track of current 'state' or changing data. This also avoids having to scrape the same data from the html repeatedly as it is always there in the state container as long is it is kept up to date.

No screenshot available. See line 20.

Problem

If the value in the number input was completely erased, the flow rate for that nozzle would display 'NaN'. This also showed in the total flow.

Solved

Using a ternary statement, I altered the "updateSingleFlow" function so that if the input value was blank the value for the amount would equal zero.

No screenshot available. See line 56.

errors.js

Problems

The ‘addKeydownError’ function was firing when entering any key including numerical values and backspace

Solved

I entered a ‘console.log(e.code)’ into the function. When I viewed what the key code was I saw that numerical keys were named ‘Digit1’ or ‘Digit9’. The regular expression I was using was only looking for the numerical values. I adjusted the regular expression to specify a numerical value preceded by ‘Digit’. I also adjusted the regular expression to include ‘Backspace’. After these changes the error function only fired when the key pressed was a non numerical key or not Backspace

No screenshot available. See line 37.

Social media

Problem

Facebook scripts wouldn't load in Mozilla firefox giving error : CORS request did not succeed.

Solved

I looked into the meaning of this error in MDN web docs and found that one possible cause was browser plugins. I realised I had a browser plugin installed which stops facebook tracking my activity. Once I disabled this plugin the script worked fine. This was the only bug I encountered with social media integration.