

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.

original spreadsheets



Rainbird 3500 series

Flow data for the 3500 series.



Planning

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

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!

Debugging

Although I didn't write any formal tests for the application I still tested it frequently as I was coding.

prevent form submission

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 parenthesis the problem was resolved.

storing and retrieving nozzle data

Problem:

There was a large amount of nozzle data to store which must be retrieved accurately to display on the page. I wasn't sure whether to use an array (which is easier to parse) or an object (which allows the data to be retrieved according to a key and allows more accurate data retrieval).

Solved:

By parsing the html DOM node list of existing table rows and retrieving the correct data depending on the 'data-nozzle' I would ensure that the nozzle data displayed was correct. This meant using a nested object rather than an array to store the nozzle with the keys correlating to the 'data-nozzle' attribute added in the html.

retrieving data from nozzle data based on pressure selected

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.

data not getting inserted into cells

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.

Targeting flow cell from indirect sibling

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.

keeping record of current flow rate

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 as it is kept up to date.

adjust single flow function not reusable

Problem

Currently the 'adjustSingleFlow' function can not be used in isolation 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.

universal reset button

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. This worked.

enable inputs function

Problem:

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

Solved:

By 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.

CORS error

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.

Social media

As integrating social media mostly relied on scripts supplied by the platforms themselves there was no real difficulty in integrating them with the application. The only problem I had was with Cross-Origin Resource Sharing errors which is detailed in the debug log. Once this was resolved each button behaved as expected, linking to the correct platform and enabling the user to share the web page there.