## Task 3: Reflection

After receiving the technical challenge and reading that the best applicants return the challenge in 5 - 7 days, I decided that despite being given 14 days, I would make it a personal test of my abilities to return it within 7 days of starting. I wanted to prove to myself that despite having Fibromyalgia, I could work at the same rate as others.

It was late afternoon when I received it and I spent the rest of the day, thinking about my current skills, which parts seemed to be within my current ability and which parts would require more research. This meant by the time I was at my computer I had a general idea of how I would approach task 1 and how I could possibly enhance the final validation app with accessibility features I noticed had not been mentioned in the user stories. I had also taken note of the simplicity of the testing suite and came to the conclusion I would approach the project as if I was building something that would be used by many in real life.

Working from the user stories was a great experience for me, I was able to read over each need and check for any overlap, this occurred the most with the CTO and Commercial Director. I realised was able to consolidate their needs at the same time, which allowed me to build a simple road map for the order of build. I broke it down into the following steps:

* HTML5 - use semantic html - framework also gives me something to work with for the JavaScript I would write
* JavaScript - name field - use regex
* JavaScript - card field - use Luhn algorithm
* JavaScript - email field - use regex
* JavaScript - submit as an email - use ‘mailto:’
* CSS - layout and design - mobile first/accessibility

This was broken down based on what I knew and what I would have to learn as I went along. I chose to start the JavaScript after having the html done as I knew the functionality of the app was most important, but would also take me the longest time to get right. I did all my work using visual studio code and my browsers dev tools.

I continued to check back with the user stories throughout the development process, in order to tailor my development efforts to meet their needs. I kept checking back to ensure I had addressed all the essential feature and functionality required.

I also used the test cases in the same way, having them before starting development allowed me to verify the functionality incrementally. It also meant I was able to ensure each part of the form met the expected criteria.

I was able to use the testing suite to safeguard against errors and invalid data, in fact I was even able to notice and safeguard against a few more commonly found in real life in terms of name and email address. It made the development process a smooth a d focused endeavour. Having both the user stories and the testing suite served me as both a guide and a checklist to follow

I followed the plan I created sequentially starting with the html, I went with best practice and used semantic html including the use of the fieldset, legend elements, around the form elements.

This would also ensure assistive technology would follow a logical journey through the form without using any aria attributes or roles. Once I was happy with this framework, I started writing the code for the name field.

Building the regex for the name field allowed me to test what I had learned through freecodecamp.org. I refreshed my memory using notes I had made during the course and was able to build a simple regex pattern that passed all the test data. It was while I was at this point in development that I realised test driven development was allowing me to know how the page should perform without seeing a working example first. I had no experience with SQL injection before the challenge and it was while I was looking for a pattern to avoid this that I realised I could also get the pattern to accept other family names like O’Connor, so I did.

After writing the function for the name field, I worked on the card field. The Luhn algorithm for checksum was completely new to me. I got lucky with the C++ example code being given along with the steps the JavaScript needed to follow. I have no more than a passing familiarity with C# however, since I have been planning to learn for some time, once my proficiency in JavaScript was high enough. I know some fundamentals such as strict typing and the data types the language declares in the script. I was able to leverage what I know to read the C++ code.

Once I was able to match the steps given to the lines of code, I had a general idea how to write the JavaScript to match. I also realised it would be best to use a regex to check the length and type of input before passing it to the Luhn algorithm. I ended up using the example first, by using quantity specifiers in the regex, it was simple to change from 16 characters to 11 characters and test my code against the number given in the example. This was also because it told me what the sum of the digits were, which is something I could check in a console.log.

Where it came to handling the two-digit numbers, I had the most trouble, but I kept researching and managed to find 3 different ways to implement it, but I thought the way that seemed most self-explanatory was to use toString() method in order to use split(‘ ‘). I then converted the string to an array of number characters using map(Number) method, which allowed me to use the array method reduce() to sum everything in the array.

I’m not completely happy with the function I came up with to validate the credit cards. If it were completely like the C++ version, I would have written code to validate separately, but I just don’t have the necessary knowledge or confidence to break up what I’ve written into two functions like that. Going forward that is something I will make sure I learn as I always want my code to be efficient and clean and the function, I wrote strikes me a bit clunky due to repetition. But the feeling I got when the test data and example number was finally validated was priceless, and I learned so much from this experience.

The function to validate the email field was also built using regex patterns to match most real-life email addresses. It was while I was researching the considerations of email regex patterns, that I was reminded of subdomains and those used for schools or government. All of this went to inform the regex pattern I built.

The final JavaScript function I wrote was for the submit event. I have created submit buttons, but I’ve never given them the functionality that the challenge was asking for. The hint to use mailto: was just what I needed, and I went straight to MDN to work out how to accomplish this without an anchor element for a hyperlink. It took a lot of reading and trial and error, especially with the mailto: URL scheme string I needed to build. I think the hardest part of this was including a final validation check to the function, because I was validating within each function for the field, I had not included returns. Once I did, the submit validation fell into place, but I didn’t like that I was now returning from all the validation functions, but they were still handling the validation checks.

It was a great feeling seeing my desktop email client open, and certainly worth all the effort.

It was after the function for the submit event was working, that I decided to add a text alert to users that there has been an error in input that I came up with the idea to create an easily updated object to store invalid input messages to alert the user to their error. I decided to use this with functions to display the message to the user and to remove the message once the input became valid. I thought that as well as highlighting the border of the input, a message in the same colour as the submit button and using the font family and font- size asked for in the wireframe. I added these to be called validation functions and the submit function on the appropriate conditional statement branches.

Creating the messages to update the user increased my knowledge of aria and the describedby attributes usage. I did need to update the functions to account for screen reader technology, by using setAttribute() method on the element created when the input is invalid.

I added each event listener for the necessary browser event prior to writing the functions. I also used console.log extensively. I apologise if I left in any unnecessary ones. I also made sure to structure the code so all event listeners are together and all validation functions are together with the error code written first. I also made sure to comment throughout my code, something I always do in order to understand it in the future and for any developers who read my code.

The final step was the CSS. It was mobile first built, using the emulator in dev tools, I was able to style the validation form to match the wireframe. With the addition of the legend for screen reader technology it was necessary for me to research how to hide it visually. I was able to find an article on the CSS Tricks website which allowed me to do this following best practice using a ‘sr-only’ class.

It was also necessary to style the invalid input messages I had added. I used flexbox in order for the form to stretch to accommodate the dynamic content and created a media query for desktop styling.

Task 1 of the technical challenge was incredibly fun; I was able to enhance skills I already had and gained new ones.

The Luhn checksum algorithm, which while frustrating to convert to JavaScript, I found it to be a challenging, enjoyable and rewarding learning experience. I’m always happy to increase my proficiency in JavaScript

I have also been able to deepen my understanding of regular expressions, to the point I can craft my own patterns for specific criteria

Accessibility is very important to me, so the skills I gained from this project have been a boon for me. Such as learning, that by using the setAttribute method and creating associative IDs for error messages I could create something that would be read by assistive technology in real time. I’ve also learned how to hide content visually but not from screen readers in a way that meets best practice and web accessibility

If I could do anything differently it would be to refactor and refine my functions, into something less DRY, but I understand that will come with time and continued learning. If I knew more about the Object-Oriented Design paradigm, I believe I would have written better functions. I also regret not knowing more about code optimisation and performance such as ‘Big O’ as I would have used this to make sure I didn’t end up with so much DRY. I look forward to learning in the future how best to decrease redundancy seen in my code.

I also made an assumption for this project, based on an imagined user journey, to add the invalid input messages. Had this been a real project, I would have gotten confirmation from either the Marketing Director directly or from my supervisor, whether to implement this feature.