**Matthew Barker – Software Development Challenge Reflection**

**How was working from User Stories? -**

I find User Stories very handy as they’re (usually) very succinct statements on what the end-product should look like and do. These ones were perfect as they stated what the code should do, but left it fairly open-ended as to how I should go about getting there.

**How was it having the test cases before you started development? -**

I found this incredibly useful as it meant I knew exactly what the code was meant to do and, perhaps more importantly, what it was *not* meant to do. Knowing the tests that would be used on my code helped clarify how deeply I would need to think about the validation, and if there were any obscure edge-cases I would need to consider. For example, on the Name validation, I decided that a name like “Martin Luther King” would be acceptable as it contains a both a given and a family name, which I took to be the bare minimum and, if the user decides to give a middle name as well, that shouldn’t then fail the validation.

**How did you use Generative AI? -**

I didn’t, I thought it would be more beneficial to my learning to take on the challenge using my own coding, problem-solving, and googling skills!

**What are the risks of using Generative AI? -**

I’ve never used generative AI before, however I would imagine the risks would be not being able to explain your code, not considering edge-case scenarios that could break your code, and potentially having trouble maintaining said code if you didn’t write it.

**How did you implement the code?**

Before writing any code I used pen and paper to outline what the code would need to do, and then breaking down each of these steps further until I was fairly confident that I had covered everything in the user stories.

I started by getting the form to look exactly as it did in user stories as my first step, and then first testing that I could grab the data entered and have it console.logged to ensure that I could then run validation checks on it.

For the Name check, I used RegExr (an incredibly useful website!) to hone in on the perfect regular expression to check for valid names. Originally, I was going to check that the given name and family name were both of at least length 2 before checking their validity, but realised that this could also be implemented in my regex to cut down on the size of the function. I considered hyphenated names as well and accounted for these in my regex, so names like Jean-Claude Van Damme would pass.

For the Email check, I luckily already had a regex check for this from my time on the Software Development course, so this was relatively quick to finish; I just had to check the email was of a valid format and then it was done!

For the Card check, strange as it sounds, I really enjoyed implementing my own function for the Luhn formula; I saw that one had been provided in another programming language, but I wanted to see if I could do it myself from scratch (I appreciate in the real-world this is not always the best approach, but I wanted my code to be my own and to know exactly what was happening and when).

Though it wasn’t mentioned in the brief, I decided to add error messages when any of the validation checks failed, to inform the user of what they needed to correct before the form could be submitted.

Finally, I abstracted all of my functions to their own files for ease of maintenance, debugging, and reviewing.

**What have you learned? What would you do differently?**

I learned how credit card numbers are validated, which was fascinating! I had never heard of the Luhn algorithm before, so that was an interesting fact to take away. I also strengthened my regex skills through trial and error and testing various names to make sure that only valid ones would pass.

If I were to undertake this challenge again, I would test it using a wider variety of email providers to ensure that it worked on as many as possible. I am a little concerned as, using a Linux virtual machine to code, the default email provider was Mozilla Thunderbird. This, however, didn’t take my specified body when using mailto: and insert it into the generated emal. I did however test the code using Gmail and the body did appear in the generated email, so I’m hoping it also does when you review it, and this is just a shortcoming of Thunderbird (which I don’t know a single person who uses it...).

The only real issue I encountered was completely forgetting that splitting the card number into an array of the individual numbers would convert each number to a string value. However, when summing the numbers and seeing that it came to ‘4111111111111111’ I quickly realised what had happened. I was a little concerned though as, in the tests, it mentions that the above number should be a valid card number, but when I put this into my function it gives a total of 27 (4+2+1+2+1+2+1+2+1+2+1+2+1+2+1+2) which is not valid.

I would also maybe be a little more critical in my Name check regex as, as it stands, a name like “James Timothy Bob Tommy Billy Smith” would pass. However, there was no mention of a name-length limit, only a minimum of given and family name (no initials), so I considered the above name to be okay (lucky for James).

Even if I am unsuccessful in securing an apprenticeship, I thoroughly enjoyed this challenge. I love logical thinking, problem-solving and using my knowledge to create something. I thought of the whole exercise as one big riddle to be solved step-by-step using my skills, and I thank you for the opportunity.