Overview

This project was created as part of a technical assessment with the requirement being to create a web page that converted numerical input into words and passed those words as a string output parameter. To do this it needed to communicate with a backend server, where the string input was passed to it and then a string was returned for output.

Frontend

For the frontend, I determined that it didn't need to be anything particularly fancy, seeing as the position is for a primarily backend role. Nevertheless, I still ensured that it conveyed sufficient information about what it was and what it did. I provided a Title, an input box with an indication of what should go in it, and then a button saying what it would do. I did decide to make the input box accept any text as opposed to restricting it to the 'number' type. The main reason was that I wanted this to be able to handle extremely large numbers and, when inputting those, it can be useful to include commas to break it up (e.g. 593,291,121,816). While this did cause a little bit of extra work on the backend, I chose to allow for commas.

For the JavaScript side, there was no need to make it complicated so I set up a simple fetch request that would send the input, wait for the output, and then modify the result field of the HTML based on what was received, displaying either the requested output or any error that occurred. As for specific decisions I made, I chose to get the input form from the HTML via its ID. I could have simply grabbed the first form on the page which would have also worked, however if this project ended up being expanded then that could have created unnecessary extra work changing it.

Backend

I used Asp.Net for the backend as I am very familiar with C# and it was the quickest and simplest way to setup a project that needed to talk to a frontend. The interaction between the two ends is handled in a controller class whereas the conversion logic is handled by a separate class. For the controller, I decided that it would confirm the input was valid and useable before sending it to the converter as it made sense to me for the converter to only worry about converting and not about any issues in the input. The controller also handles any exceptions that occur in the conversion process and then informs the frontend that there was a problem. Again, I could have had the converter handle its own exceptions, but then communicating that with the frontend would have been more complicated so it made more sense for me to let the controller handle it.

I allowed there to be a difference between valid input from the frontend and valid input into the converter. For instance, -005,200.0500 is a valid input, but would cause problems for the converter. So, the controller sanitises the input in order to convert it into something the converter can use (in this case, 5200.05). While I could have just rejected this as invalid input, I decided that cleaning it up would allow for more natural input from the user. The sanitisation does the following:

-Remove spaces (2 05 -> 205)

-Removes commas (200,000 -> 200000)

-Removes excess zeroes (097.110 -> 97.11)

-Adds missing zero (.052 -> 0.052)

The number to words conversion is implemented in a static class and has two methods of handling numbers based on their size. For any number that surpasses the highest number name given (in this case decillion), the number is converted into significantly simplified text (five point three times ten to the power of forty). Whereas for numbers within the accepted range (again, up to 999.999... decillion) the conversion process will output the entire number. For a quick overview of how this happens, the number is first split between the whole number and the decimal. Then the whole number is split up into 3-digit groups, starting from the end and working backwards (21592192 -> 21 592 192). Then it works through these groups, converts them into text, and gives them the relevant name afterwards (million, billion, etc). The way the number groups are converted is different for the 1st digit and the last 2 digits. The 1st is simply directly converted (1-one, 2-two, etc) whereas the last 2 need to be handled based on numerous factors (14 needs to be converted together whereas 24 needs to be separated as 20 and 4). After this, the decimals are handled by just directly converting each digit to its equivalent name (.52 -> point five two).

The reason I chose to split the number into these groups is that, regardless of the number size, it will still be fast, since any truly enormous number is handled in a simpler way. Additionally, this method is very easy to debug and doesn't require much overhead. As for why I chose this solution over others, I deemed this one to be the simplest to implement and there were minimal downsides given the scope of this project.

As for why I chose to handle the 2-digit numbers the way I did (splitting 84 into 80 and 4), this is primarily due to personal preference. While hardcoding the conversions for all 100 possibilities (00-99) was an option, I prefer to avoid hardcoding things wherever possible. As such, I only wrote in the conversions necessary (1-19, 20,30...90). Writing the 100 possibilities would have simplified the code but, ultimately wouldn't have made much of a difference in relation to the code's performance.

Limitations

While my solution accomplishes the requirements and handles numerous edge cases, there are still limitations which have not been accounted for.

Firstly, while my solution can handle incredibly large numbers, it stops providing complete output upon reaching the undecillions (1e36). After this point it can still accommodate incredibly large numbers however it does eventually start to lag and finally crash if the numbers get absurdly large (it was able to get above 1e1000000).

Secondly, my solution cannot handle different languages. If I wanted it to do this then I would have needed to setup a simple localisation system. I did not do this as the user is expected to be using English however it is important to note should the project ever be given to non-English speakers.

Lastly, my solution is unable to handle different number formats. Some countries swap commas and decimals (200.192,05 = 200,192.05) and this system would consider that invalid input. Additionally, it cannot handle scientific notation in either of its formats (1e12 vs 1x10^12). These would also be considered invalid input.

Assumptions

Since the assessments requirements were very simple, there were several assumptions I had to make in order to create this project. These assumptions were:

1. Handling large numbers

Since no upper limit was given for the numbers that can be provided, I had to decide how large of a number should be handled normally and then what to do with numbers larger than that. I chose to allow up to 999...9 decillion and then use simplified language for higher numbers.

2. Invalid inputs

Both the handling of invalid input and what constitutes invalid input were not specified. I decided to allow for all kinds of numbers (very big, very small, negative) and also to allow the user to input commas as they are often used to make large numbers more readable (39418953 -> 39,418,953). As for handling invalid input, I decided to simply inform the user that their input was invalid. Something I could have done would be to inform them of what was invalid about their input, but I deemed that unnecessary for this project.

3. Negative numbers

While the example did not mention negative numbers, I always assumed they were expected to be handled. However, I did need to decide how they would be referred to, as both "Negative" and "Minus" are valid ways of saying a negative number (Minus Two vs Negative Two). I decided to go with "Negative" but this is primarily a personal preference as they can be used interchangeably when speaking.

4. Frontend's purpose

I made the assumption that, given that the position is a C# developer, that the purpose of creating the frontend is primarily to interact with the backend. Due to this assumption, I made the frontend very simple with minimal styling, seeking only to provide an interface with the backend.

5. Dollars or Numbers

This aspect of the assessment was confusing to me. The assessment requests that I convert numerical input into words (e.g. 5 -> five) however the example given converts it into currency (5 -> five dollars). While I sent in a clarification request, I did not want to avoid working on this while waiting for an answer. As such, I decided to go with the assessment's description as opposed to its example, converting the numbers into words and not currency. My reasoning for this is that, should I be mistaken and receive clarification, modifying the code will be simple, as adding text for Dollar(s) and Cent(s), as well as making the slight formatting changes, would not take very long.