Justification:

Looking at this problem, I quickly identified what the patterns were in the way a number is written. The two lowest order digits in a number (the tens and the ones) have a unique and variable form of verbal expression. When together they are higher than nine and less than twenty, the form a single word depending on both digits (eg eleven or nineteen). However, when they are above this range they follow a pattern where a word for the number In the tens place is hyphenated with a word for the number in the ones place (eg forty-five), and when they are below this range they are expressed with just the name of the digit in the ones place (eg two). This pattern does not just appear in the tens and ones spaces of a number, it also repeats after the 3rd digit, and appears when expressing the hundredths of a number (represented in the first two decimal places). As such, it felt obvious that determining this ought to be its own function.

The 3rd digit in a number, the hundredths space, is always represented with the number in that slot followed by "hundred and." This, followed by tens and ones, would form a string like "One hundred and forty-five" for example. There are cases where this does not apply, but those can be handled with some simple checking of criteria. This pattern is repeated in every 3 digits in a number, separated by an indicator of the order of magnitude, such as thousand or million. From this it became apparent that the same operation could be repeated with some iteration on magnitude for a number of any length. As such, I wrote a function that accepts a string containing an integer, generates the verbal description of the last 3 digits (eg "one hundred and forty-five"), adds that to a string, and then calls itself again to do the same for the rest of the integer string, adding the incremental order of magnitude between each 3 digit slice, until the next integer string is empty and a phrase is been fully constructed.

After this, the number of cents is determined using the first function described and appended to the end of the string, and the string is returned. It seemed to me that this approach was the most versatile, and could be applied to a number of any length, as long as there is a name for it.

There is a lot of somewhat inefficient edge case checking occurring in my code, so I decided to sacrifice some memory efficiency in my retrieval of the words representing numbers, by storing all of those words in a list of strings, which the program could read from using the index of the number representing each word.

My solution uses a Dotnet Blazor Web App, which I chose because it is quick and easy, and I didn't think the architecture of the app was as important as the functions that operate it for the purposes of this tech demo.

There exists a version of this that takes place in one big for loop which may have streamlined some of the edge case handling, such as the number 1,000,000,001, or one billion and one. My solution checks after every 3 digit slice whether or not a thousand,

or million, needs to be inserted by checking if the overall value of the 3 digit slice is zero. Something that iterated over each digit and held something in memory that indicated what extra words were needed may have had an easier time with this, but I think it would have been a little less efficient, and much less readable.