Just another Frontend Developer test

Ciao!
Quello che troverai non è il solito test frontend, quello a cui puntiamo nel nostro team è avere persone che sappiano ragionare, il resto è tutto in discesa :)

COSA DEVI FARE
Risolvere la traccia che troverai di seguito, scrivendone anche i test (anche questi troverai di seguito) in modo che siano verificate tutte le casistiche. Inoltre ti chiediamo di creare una veloce interfaccia web per permettere anche noi di testare la soluzione da te trovata.

OUTPUT
Decidi tu come inviarci il lavoro: zip per email, repo git o qualsiasi altro modo.

COSA DEVI UTILIZZARE

- NodeJS
- ReactJS
- Sei libero di approfondire come vuoi il test, una veloce UI per utilizzarlo? Docker? Sentiti confident di arricchire il tuo lavoro come meglio credi.

COME TI VALUTIAMO

- Semantica, organizzazione e documentazione js;
- Affidabilità e performance;
- Iniziativa e inventiva;
- Senso estetico (non farai il designer ma ci piacerebbe un po' di senso estetico);
- Usabilità Mobile e Desktop, ovviamente cross browser (IE11+ per non farti impazzire);

Ti ringraziamo in anticipo per il tuo tempo e ti promettiamo che riceverai un feedback approfondito sul tuo esercizio. **Non ti preoccupare se non riesci a concludere al 100% il test, valutiamo la logica che applicherai non facciamo test binari da "dentro" o "fuori" :)**

TRACCIA:

Please create a function which takes a number and returns the name of the number in English as a lowercase string. The function should support at least the numbers ±2 Quadrillion (±2,000,000,000,000,000) as well as positive and negative infinity.

Rules

- **positive integers -** Print number in english.
  - Numbers between 20 and 99 use hyphens. e.g. forty-five
  - Use a space to separate all other words. e.g. one hundred twenty-three

    - The word 'and' is used to separate the tens space from the hundreds space in each period
        - 111 -> Good: "one hundred and eleven" - Bad: "one hundred eleven"
        - 1,101,101 Good: "one million one hundred and one thousand one hundred and one" - Bad: "one million one hundred one thousand one hundred one"
    - ...also the word 'and' is used to separate the tens and ones space from the lowest number the left of the tens place for numbers over 1000.
        - 1,001 -> Good: "one thousand and one" - Bad: "one thousand one"
        - 1,000,001 -> Good: "one million and one" - Bad: "one million one"
        - 1,000,011 -> Good: "one million and eleven" - Bad: "one million eleven"
        - 1,000,111 -> Good: "one million one hundred and eleven" - Bad: "one million and one hundred and eleven"
        - 1,011,011 -> Good: "one million eleven thousand and eleven" Bad: "one million and eleven thousand and eleven"
    - Support integers up to ±2 Quadrillion.
    - Consult this list for large number names. Always use the 'short scale' popular in the USA.
- **Zero/nil -** Print 'zero'.
- **Negative integers** - Print 'negative' before the number
- **Decimal numbers** - Print the number to the left of the decimal then 'point' then the numbers to the right of the decimal each digit at a time. (see example)
    - Support at least 5 decimal places.
- **Non-numbers -** Strings that evaluate to numbers should be converted to numbers. NaN values must throw an error.
- **Positive / Negative infinity -** Print 'infinity' or 'negative infinity'

EXAMPLE:

```
numberToEnglish(1); // -> "one"
numberToEnglish(11); // -> "eleven"
numberToEnglish(1.23); // -> "one point two three"
numberToEnglish(-45); // -> "negative forty-five"
numberToEnglish(100023999); // -> "one hundred million twenty-three thousand
nine hundred and ninety-nine"
```

TEST:

```
Test.assertEquals(numberToEnglish(1), "one", "1 -> one");
Test.assertEquals(numberToEnglish(5), "five", "5 -> five");
Test.assertEquals(numberToEnglish(10), "ten", "10 -> ten");
Test.assertEquals(numberToEnglish(11), "eleven", "11 -> eleven");
Test.assertEquals(numberToEnglish(12), "twelve", "12 -> twelve");
Test.assertEquals(numberToEnglish(18), "eighteen", "18 -> eighteen");
Test.assertEquals(numberToEnglish(20), "twenty", "20 -> twenty");
Test.assertEquals(numberToEnglish(19000), "nineteen thousand", "19000 ->
nineteen thousand");
Test.assertEquals(numberToEnglish(319000), "three hundred and nineteen
thousand", "319000 -> don't forget the 'and'");
Test.assertEquals(numberToEnglish(1000000), "one million", "1000000 -> one
million");
Test.assertEquals(numberToEnglish(1000001), "one million and one", "1000001
-> one million and one");
Test.assertEquals(numberToEnglish(1011011), "one million eleven thousand and
eleven", "1011011 -> one million eleven thousand and eleven");
Test.assertEquals(numberToEnglish(1101101), "one million one hundred and one
thousand one hundred and one", "all the ands");
Test.assertEquals(numberToEnglish(-6000006), "negative six million and six");
Test.assertEquals(numberToEnglish(100023999), "one hundred million
twenty-three thousand nine hundred and ninety-nine");
Test.assertEquals(numberToEnglish(3.14159), "three point one four one five
nine", "Decimal numbers count each digit");
Test.assertEquals(numberToEnglish(0.0001), "zero point zero zero zero one",
"Include leading zeroes in decimals");
Test.assertEquals(numberToEnglish(-65721.55531), "negative sixty-five
thousand seven hundred and twenty-one point five five five three one");
Test.assertEquals(numberToEnglish(0), "zero", "0 -> zero");
Test.assertEquals(numberToEnglish("6"), "six", "strings that evaluate to
numbers are ok");
Test.assertEquals(numberToEnglish(Number.POSITIVE_INFINITY), "infinity",
"positive infinity");
Test.assertEquals(numberToEnglish(Number.NEGATIVE_INFINITY), "negative
infinity", "negative infinity");
Test.assertEquals(numberToEnglish(-50), "negative fifty", "Negative numbers
should include the word 'negative' before the first digit.");
Test.assertEquals(numberToEnglish(-1234567899), "negative one billion two
hundred and thirty-four million five hundred and sixty-seven thousand eight
hundred and ninety-nine", "1,234,567,899");
Test.expectError("NaN should throw an error.", function () {
numberToEnglish(NaN) });
```