NodeJs Pre-Interview Coding Test

Welcome to the Reckon coding test!

Build the code as if this was just one of many tasks that you have to do in one day, using whatever libraries / frameworks you feel comfortable with unless explicitly instructed not to. The tasks are fairly straightforward and should take no longer than an hour, there are no major "gotchas" but there will be input and output that exist beyond the samples provided. When finished, please create a repository and push the unzipped, raw source to a public bitbucket / github repository and then submit a link to this repository. Please submit working nodeJs source code to solve the problem along with any supporting code that you might have used in testing.

Test 1:

We have two api endpoints

Endpoint 1:

https://join.reckon.com/test1/rangeInfo provides an upper and lower bound of numbers

An example response may be

```
{
    "lower": 1,
    "upper": 100
}
```

`Endpoint 2:`

https://join.reckon.com/test1/divisorInfo provides a list of outputs that we want to check for .

```
{
    "outputDetails": [
        {
            "divisor": 3,
            "output": "Boss"
        },
        {
            "divisor": 5,
            "output": "Hogg"
        }
    ]
}
```

Requirements

Given the two endpoints provided, provide nodeJs website running on http://localhost:9999/ .

GIven this website, when the user visits the root url, the backend calls *Endpoint1.* For all numbers inclusive of the lower and upper bound, go through and check if it divisible by each of the divisors provided in *Endpoint2.*

If the number is wholly divisibele , log the output as a result to the browser. If multiple outputs are satisfied, print outputs that are satisfied.

The api endpoints provided can be quite unreliable, if it's the case that one of the endpoints fail, have the code try again until successful results are returned.

Expected Output

Using the example output from the two endpoints above, we would expect to see the output writen to the user's browser:

Sample Output

```
1:
2:
3: Boss
4:
5: Hogg
6: Boss
7:
8:
9: Boss
10: Hogg
11:
12: Boss
13: Boss
14:
15: BossHogg

[....etc....]
```

Test 2

Preface:

Do **NOT** use any extended functionality or packages of javascript to solve this problem. (For example, don't use the string finding methods of subs tring, indexof, lastIndexOf, match, search, split or any npm modules related to string / text processing).

Importantly, the samples listed below should not be considered the only inputs and outputs.

Requirements

We need an api endpoint that will find all the occurrences of a particular set of characters in a string and post the results back to another api.

We have two endpoints available that provide the text, and the subTexts to search for.

The api endpoints provided can be quite unreliable, if it's the case that one of the endpoints fail, have the code try again until successful results are returned.

Endpoint1: https://join.reckon.com/test2/textToSearch provides the text to search in. An example output might be:

```
{
    "text": "Peter told me (actually he slurrred) that peter the pickle
piper piped a pitted pickle before he petered out. Phew!"
}
```

Endpoint2: https://join.reckon.com/test2/subTexts provides a list of subTexts that we need to search the output of Endpoint1 for. An example output might be

```
{
    "subTexts": [
        "Peter",
        "peter",
        "Pick",
        "Pi",
        "Z"
    ]
}
```

- The solution should match the subtexts against the text , outputting the positions of the beginning of each match for the subtext within the textToSearch.
- The set of characters can occur anywhere within the string.
- Multiple matches are possible
- Matching is case insensitive
- If no matches have been found, “<No Output>” is generated
- Your api endpoint should POST the results to the endpoint at https://join.reckon.com/test2/submitResults with the results in the format listed below.
- The api endpoints provided can be quite unreliable, if it's the case that one of the endpoints fail, try them again until succesful results are returned.

Expected Output

For the sample data listed above, this is the data that is expected to be processed by our endpoint.

```
{
    "candidate" : "<your name>",
        "text: "Peter told me (actually he slurrred) that peter the
pickle piper piped a pitted pickle before he petered out. Phew!",
    "results": [
        {
            "subtext": "Peter",
            "result": "1, 43, 98"
        },
        {
```

```
            "subtext": "peter",
            "result": "1, 43, 98"
        },
        {
            "subtext": "Pick",
            "result": "53, 81"
        },
        {
            "subtext": "Pi",
            "result": "53, 60, 66, 74, 81"
        },
        {
            "subtext": "Z",
            "result": "<No Output>"
        }
    ]
}
```