

# Fiche d'investigation de fonctionnalité

<b>Fonctionnalité :</b> Recherche	<b>Fonctionnalité #2</b>
<b>Problématique :</b> Pour se démarquer de la concurrence nous voulons un algorithme de recherche rapide pour que la recherche paraisse instantanée cf. <a href="#">Figure 4 Flowchart</a>	

<b>Option 1 : approche utilisant les boucles (while, for, forof, forin)</b> cf. <a href="#">Figure.1.1 Code (loop)</a> Dans cette option la recherche se fait à travers différentes boucles successives	
<b>Avantage</b> <ul style="list-style-type: none"><li>• lisible</li><li>• facilement maintenable/modifiable</li></ul>	<b>Inconvénients</b> <ul style="list-style-type: none"><li>• très lent pour la comparaison de string</li><li>• utilise un buffer de résultat</li><li>• code plus verbeux (plus long)</li></ul>
<ul style="list-style-type: none"><li>• Score <a href="#">JsBench</a> : <b>3935/s</b></li><li>• <b>19</b> lignes ( fonction de comparaison de string inclus )</li></ul>	

<b>Option 2 : approche fonctionnelle (foreach, map, filter, find, reduce, ...)</b> cf. <a href="#">Figure 2.1 - Code (functional)</a> Dans cette option la recherche se fait grâce aux fonctions membres des prototypes d'objets js	
<b>Avantage</b> <ul style="list-style-type: none"><li>• comparaison de string très rapide</li><li>• n'utilise pas de buffer de résultat</li><li>• code plus court</li></ul>	<b>Inconvénients</b> <ul style="list-style-type: none"><li>• <code>Array.filter()</code> provoque plus d'instructions que <code>for..of</code></li><li>• moins facilement maintenable</li></ul>
<ul style="list-style-type: none"><li>• Score <a href="#">JsBench</a> : <b>20699/s</b> ( 520% de "loop" )</li><li>• <b>10</b> lignes ( 52% de "loop" )</li></ul>	

**Option 3 : approche hybrid utilisant les boucles sauf pour la comparaison de string** cf. [Figure.3.1 Code \(hybrid\)](#)

Dans cette option la recherche se fait à travers différentes boucles successives, mais la recherche dans une string se fait via la fonction `String.includes()`

**Avantage**

- beaucoup plus rapide
- facilement maintenable/modifiable

**Inconvénients**

- utilise un buffer de résultat
- code un peu moins lisible

- Score [JsBench](#) : **55751**/s ( 260% de "functional" | 14100% de "loop" )
- **12** lignes ( +20% de "functional" )

**Solution retenue :**

L'objectif étant la performance, j'ai retenue l'approche "hybrid", qui est de très loin la plus performante en plus d'être facilement maintenable.

## Annexes

	dataset x 1			dataset x 1000		
query	"coco"	"tarte"	"noresult"	"coco"	"tarte"	"noresult"
Functional	20699	29709	26643	32	31	22
Loop	3935	7313	6073	9	8	6
Hybrid	55751	80104	77884	89	88	69

Figure 0 - Benckmark [JsBench](#)

```

1 function strincludes(str, query) {
2   for (let i = 0; i < str.length; i++)
3     if (str.substring(i, i + query.length).toLowerCase() === query)
4       return true;
5   return false;
6 }
7
8 const results = [];
9 for (const recipe of recipes) {
10   const name = recipe.name.toLowerCase();
11   const description = recipe.description.toLowerCase();
12   let ingredients = '';
13   for (const ingredient of recipe.ingredients)
14     ingredients += ` ${ingredient.toLowerCase()}`;
15   if (
16     strincludes(name, query) ||
17     strincludes(description, query) ||
18     strincludes(ingredients, query)
19   )
20     results.push(recipe);
21 }

```

Figure 1.1 - Code (loop)

```
1 const results = recipes.filter(  
2   (recipe) =>  
3     recipe.name.toLowerCase().includes(query) ||  
4     recipe.description.toLowerCase().includes(query) ||  
5     recipe.ingredients  
6       .reduce((prev, curr) => prev.concat(curr.ingredient), [])  
7       .join(' ')  
8       .toLowerCase()  
9       .includes(query)  
10 );
```

Figure 2.1 - Code (functional)

```
1 const results = [];  
2 for (const recipe of recipes) {  
3   let ingredients = '';  
4   for (const ingredient of recipe.ingredients)  
5     ingredients += ` ${ingredient.ingredient.toLowerCase()} `;  
6   if (  
7     recipe.name.toLowerCase().includes(query) ||  
8     recipe.description.toLowerCase().includes(query) ||  
9     ingredients.includes(query)  
10  )  
11    results.push(recipe);  
12 }
```

Figure 3.1 - Code (hybrid)

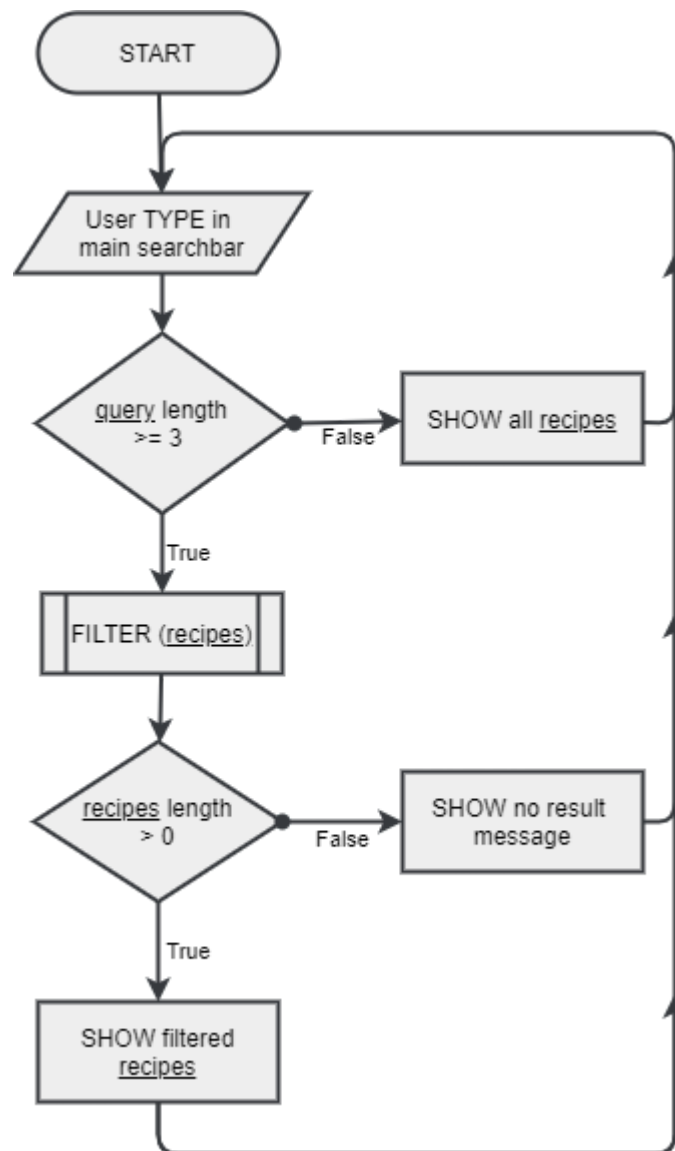


Figure 4 - Flowchart