

Fiche d'investigation de fonctionnalité

Fonctionnalité : Recherche	Fonctionnalité #2
Problématique : Pour se démarquer de la concurrence nous voulons un algorithme de recherche rapide pour que la recherche paraisse instantanée cf. Figure 4 Flowchart	

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

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

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

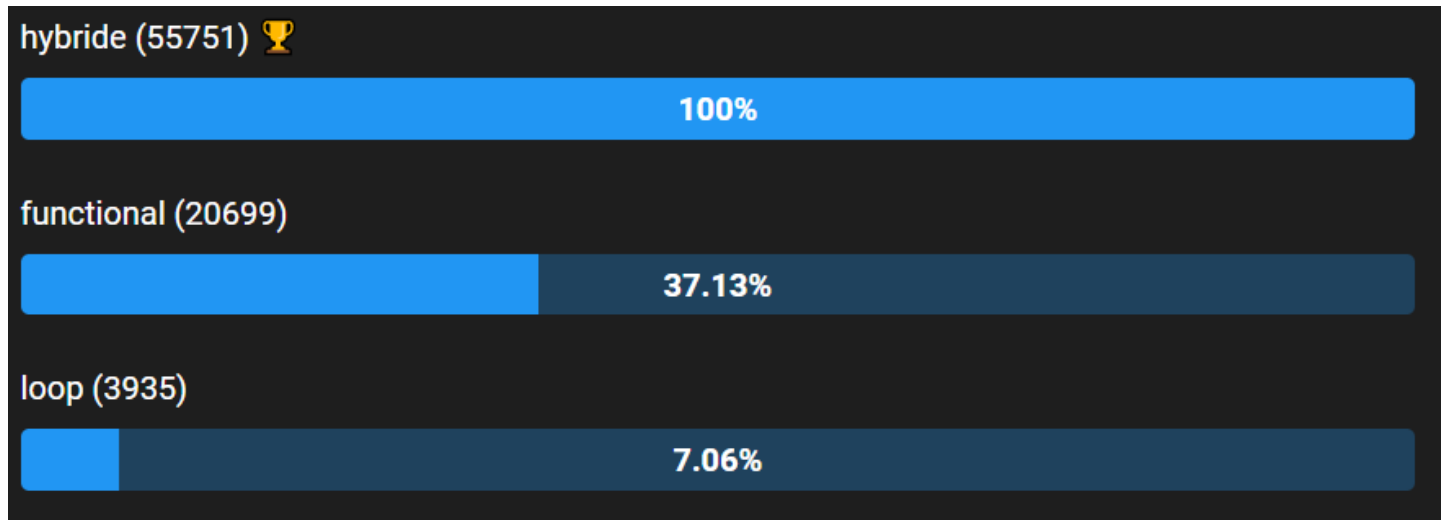


Figure 0 - Benckmark [JsBench](#)

```
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 const results = [];  
8 for (const recipe of recipes) {  
9   const name = recipe.name.toLowerCase();  
10  const description = recipe.description.toLowerCase();  
11  let ingredients = '';  
12  for (const ingredient of recipe.ingredients)  
13    ingredients += ` ${ingredient.toLowerCase()}`;  
14  if (  
15    strincludes(name, query) ||  
16    strincludes(description, query) ||  
17    strincludes(ingredients, query)  
18  )  
19    results.push(recipe);  
20 }
```

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.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)

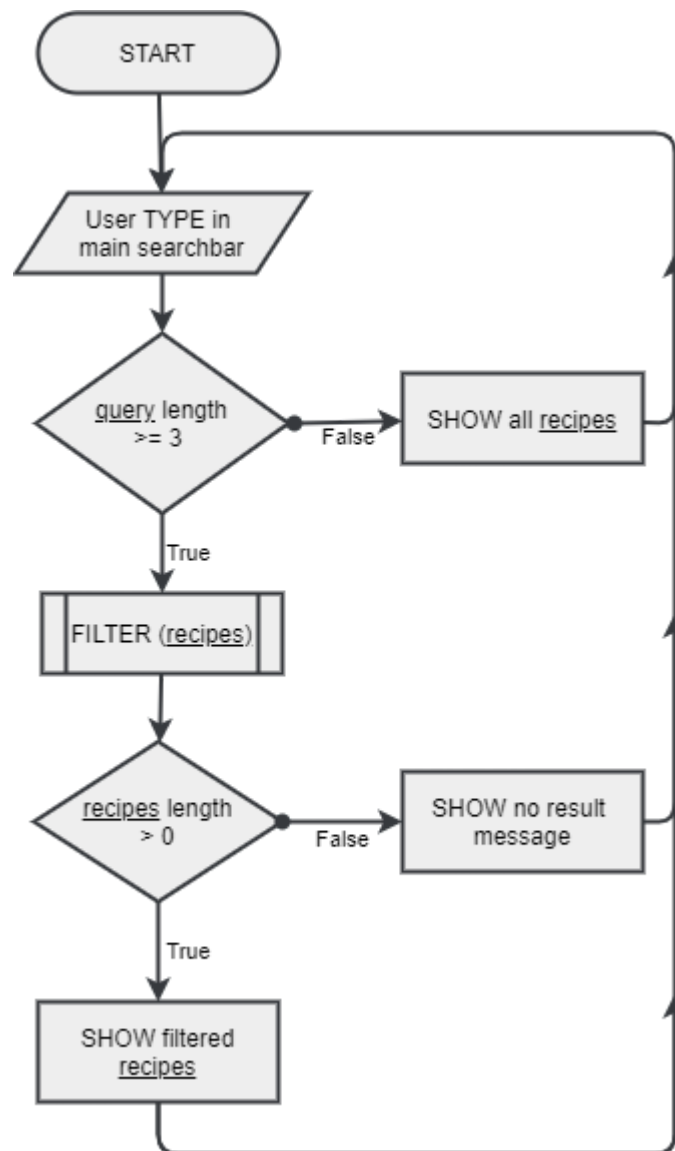


Figure 4 - Flowchart