Fiche d'investigation de fonctionnalité

Fonctionnalité : Recherche principale	**Fonctionnalité de recherche générale #1**
Problématique : Afin de pouvoir donner la meilleure expérience de recherche aux utilisateurs, nous cherchons à rendre le plus fluide possible les séquences de requêtes de la barre principale dans le but d'afficher les recettes le plus rapidement possible de manière pertinente. Le but de cette fonctionnalité est de se démarquer de la concurrence pour proposer aux utilisateurs une séquence de recherche très réactive et pertinente. En effet il est important d'avoir une recherche principale très performante pour offrir aux usagers le meilleur outil possible pour leur utilisation.	

Option 1 : On effectue un algorithme utilisant la programmation fonctionnelle avec les méthodes de l'objet array (foreach, filter, map, reduce etc..) On utilise la méthode filter pour générer un tableau de recettes.	
Avantages : - Recherche rapide, fluide et pertinente - Comptabilité avec tous les navigateurs - Moins de répétitions, moins de lignes de code - Code plus maintenable (ESLINT, structure) - Plus d'opérations dans un temps imparti	**Inconvénients :** - Lisibilité du code moins claire
Nombre de de caractères à remplir pour déclencher la recherche : 3	

Option 2 : Utilisation de boucles natives (while,for etc) Utiliser une boucle pour l'ensemble des recettes pour voir s'il y a un match avec la valeur de la recherche	
Avantages : - Lisibilité du code plus claire	**Inconvénients En cours** - moins de calculs pour le navigateur dans un temps imparti - Boucles moins performantes - Plus de lignes de codes
Nombre de de caractères à remplir pour déclencher la recherche : 3	

Solution retenue : **En cours** En testant les 2 algorithmes dans un benchmark nous nous sommes rendu compte que la solution la plus rapide et pertinente était l'option 1 du filter qui effectue plus d'opérations dans un temps imparti.

Annexes

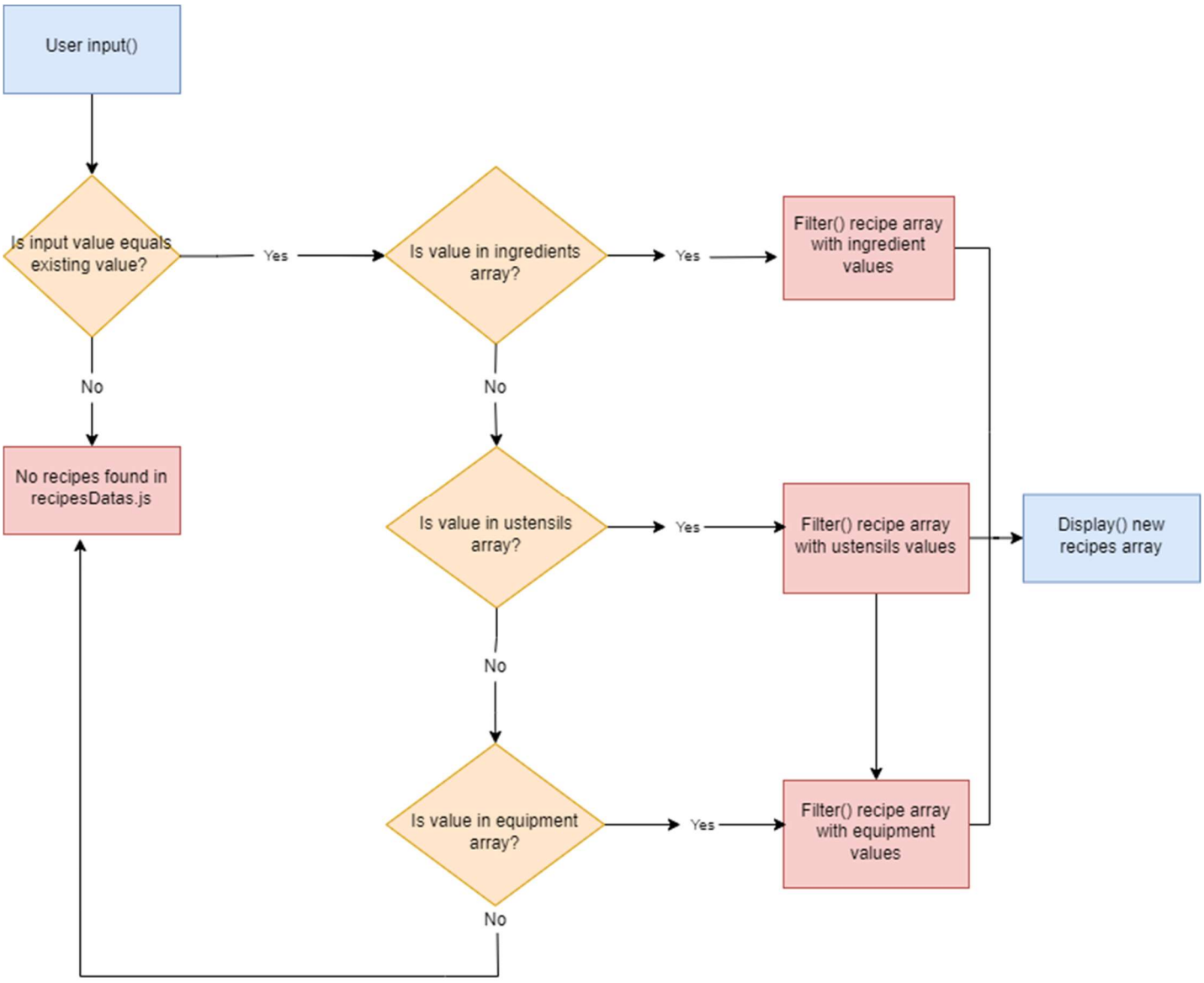


Figure 1 – Utilisation de la méthode filter pour créer un nouveau tableau si la valeur de la recherche correspond à une valeur liée aux ingrédients, ustensiles, appareils d'une recette.

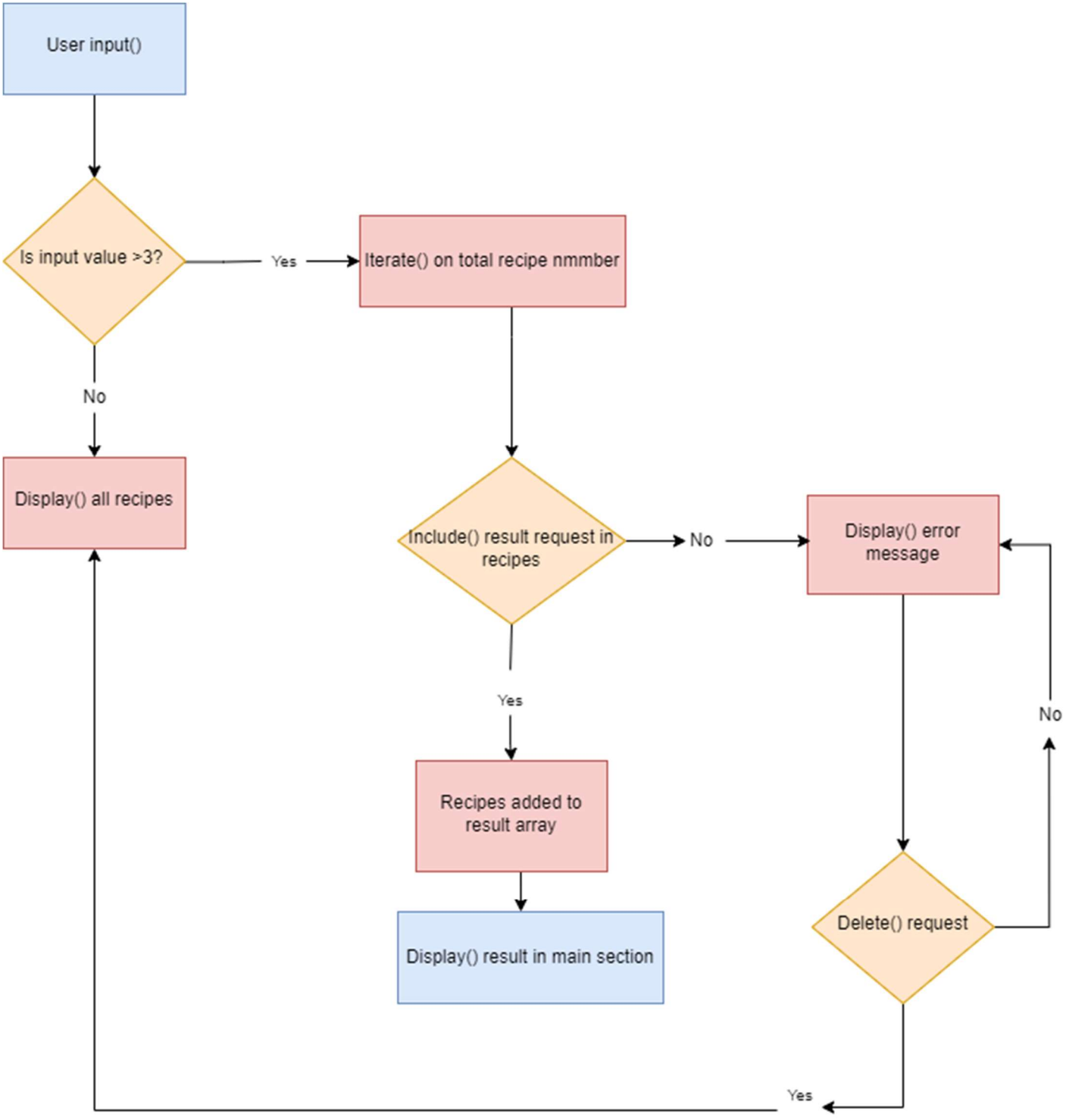


Figure 2 : Utilisation d'une boucle for sur l'ensemble des recettes : création d'un tableau vide et utilisation de la méthode push si la recette remplie une des conditions de la recherche.

Analyse des Performances JS

1. Benchmark JSBen.ch

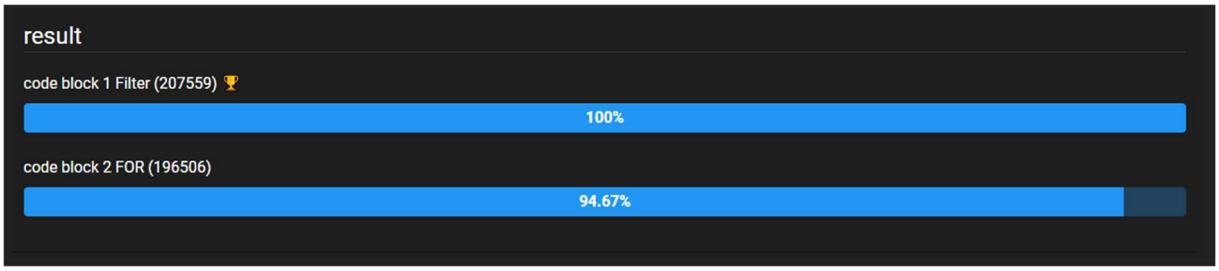
Lien : https://jsben.ch/ZD9id

On test les performances JS suivant plusieurs scénarios. Les nombres du tableau indiquent le nombre total d'opérations/seconde. Plus la valeur est grande plus l'algorithme est performant car il y a plus d'opérations par seconde dans un temps imparti :

	Algo1 Filter	Algo2 For
Scenario1 (occurrence: ex: poulet) data: 50	191823	190174
Scenario2 (occurrence: ex: "") data: 50	1561076	332013
Scenario3 (occurrence: ex: poulet) data: 500	16896	16007
Scenario4 (occurrence: ex: poulet) data: 1000	9894	9670
Scenario5 (occurrence: ex: poulet) data: 5000	6821	6612

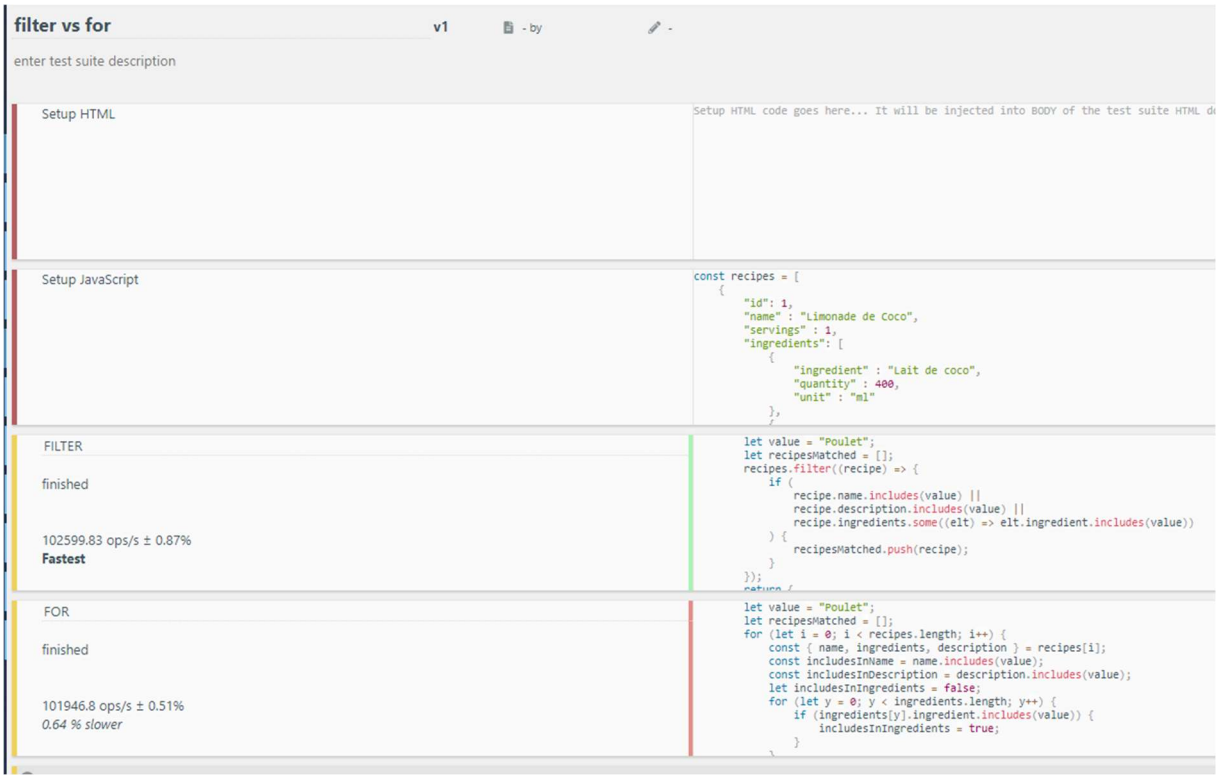
On remarque ici que l'algorithme avec Filter est plus performance que celui avec les boucles FOR.

Par exemple on obtient pour le scénatio1 avec 50 données :



Outils tiers :

https://jsben.ch/

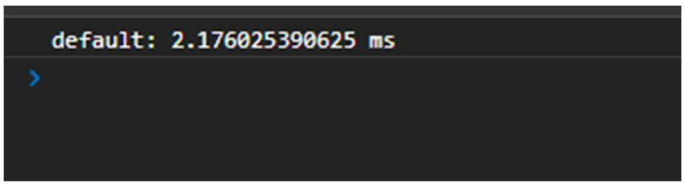


Le résultat précédent se confirme et la méthode avec filter est plus performante. On obtient plus d'opérations par secondes pour filter

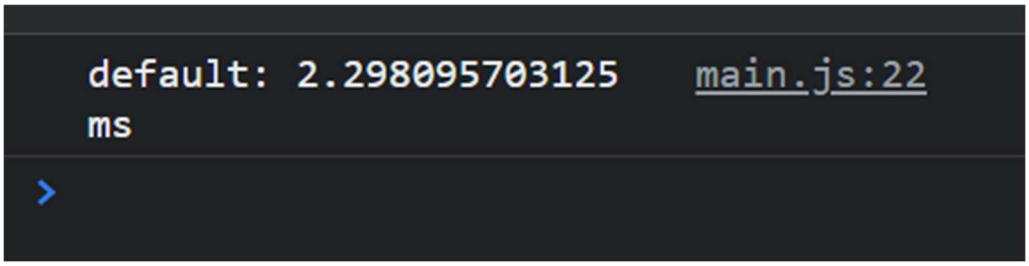
Outils internes (code JS) :

console.time(), console.timeEnd()

Avec méthode filter :



Avec la méthode for :



L'utilisation de console.time() confirme aussi une meilleure performance d' exécution plus rapide pour filter.

Liens Comparaison boucles JS (ressources outils tiers/internes) :

https://leanylabs.com/blog/js-forEach-map-reduce-vs-for-for_of/

https://stackoverflow.com/questions/1003855/how-can-i-benchmark-javascript-code

https://stackoverflow.com/questions/44612083/which-js-benchmark-site-is-correct

https://stackoverflow.com/questions/111368/how-do-you-performance-test-javascript-code