Fiche d'investigation de fonctionnalité

Fonctionnalité : Recherche dans la barre de recherche principale	**Fonctionnalité #2**
Problématique : Afin de rendre l'expérience utilisateur fluide, il nous faut développer un algorithme de recherche efficace, maintenable et rapide qui sera utilisé dans l'input de la barre de recherche principale.	

Option 1 :

Object Methods - Programmation fonctionnelle avec méthodes de l'objet Array (Fig 2, Annexe 1)

Dans cette version de l'algorithme, nous utilisons les méthodes de l'objet Array tels que .filter() and .some(). Cela vérifie si le mot renseigné dans la barre de recherche apparaît dans le nom de la recette ou tout ingrédient, en utilisant la méthode includes().

Avantages	**Inconvénients**
⊕ Lecture et maintenabilité : Code plus court et plus lisible, maintenabilité en équipe accrue ⊕ Performances : légèrement plus rapide, comme démontré par le benchmark (cf Annexe 1, Fig. 1)	⊖ Moins de contrôle sur l'exécution de l'algorithme avec ces méthodes préfabriquées : compliqué à optimiser dans des cas plus complexes ⊖ Usage de mémoire : Les fonctions telles que. filter() et .some() tendent à consommer plus de mémoire

Nombre de caractères minimum à remplir dans la barre de recherche : 3

Option 2 :

Native Loops - Boucles natives (Fig. 3, Annexe 1)

Dans cette version de l'algorithme, nous utilisons les boucles natives (for) pour itérer manuellement, et à chaque étape, à travers la liste des recettes et de leurs ingrédients.

Avantages	**Inconvénients**
⊕ Contrôle et flexibilité : Possibilité d'optimiser chaque partie de l'algorithme car nous avons le contrôle sur chaque étape. ⊕ Usage de mémoire: les boucles natives utilisent légèrement moins de mémoire.	⊖ Lecture et maintenabilité : Code plus long et moins lisible, plus compliqué à maintenir dans des projets plus complexes. ⊖ Performances : le benchmark a montré une performance très légèrement plus lente (cf Annexe 1, Fig. 1)

Nombre de caractères minimum à remplir dans la barre de recherche : 3

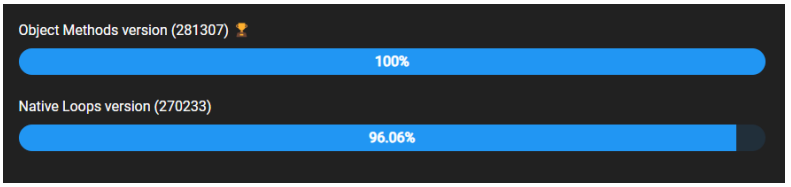
Solution retenue :
Dans le contexte de ce projet, nous avons choisi de retenir l'**option 1**, soit en utilisant la **programmation fonctionnelle avec méthodes de l'objet Array**.
Cette option offre une meilleure lisibilité, ce qui peut s'avérer crucial pour la maintenabilité de celle-ci au sein d'une équipe. De plus, malgré le fait que la meilleure performance vis-à-vis de l'autre option soit négligeable, l'efficacité et la facilité d'implémentation de cette option en font un meilleur choix dans ce cas précis.

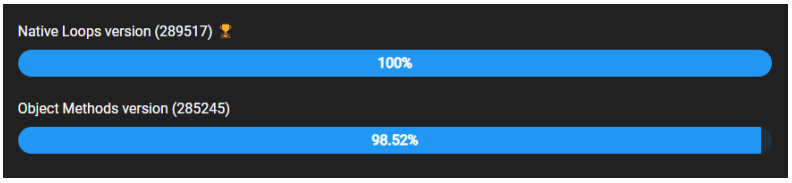
A noter que dans le cadre de projets plus complexes, il serait peut-être préférable d'opter pour l'option 2, car l'usage de la mémoire et le manque de contrôle pourraient être suffisamment pénalisants.

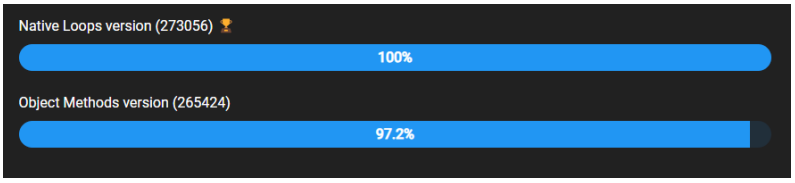
Annexe 1

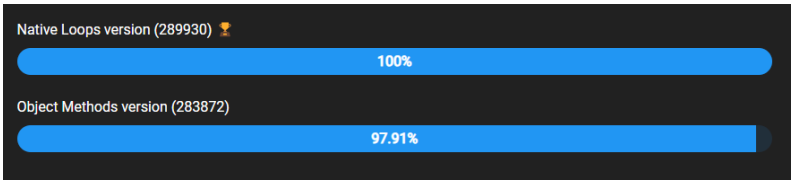
Figure 1 – Résultats des benchmarks menés sur jsben.ch

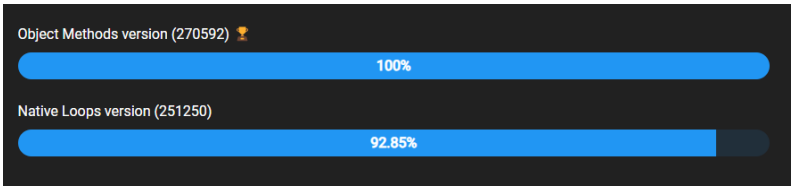
TESTS	RESULTATS

Sucre
`filterWithOM(recipes, « sucre »)`
`filterWithNL(recipes, « sucre »)`
	

Beurre
`filterWithOM(recipes, « beurre »)`
`filterWithNL(recipes, « beurre »)`
	

Menthe
`filterWithOM(recipes, « menthe »)`
`filterWithNL(recipes, « menthe »)`
	

Cuillère
`filterWithOM(recipes, « cuillère »)`
`filterWithNL(recipes, « cuillère »)`
	

Fraise
`filterWithOM(recipes, « fraise »)`
`filterWithNL(recipes, « fraise »)`
	

MOYENNES FINALES	**Version « Object Methods » : 98.72 %** **Version « Native Loops » : 97.78 %**

Figure 2 – Algorigramme – Version « Object Methods »

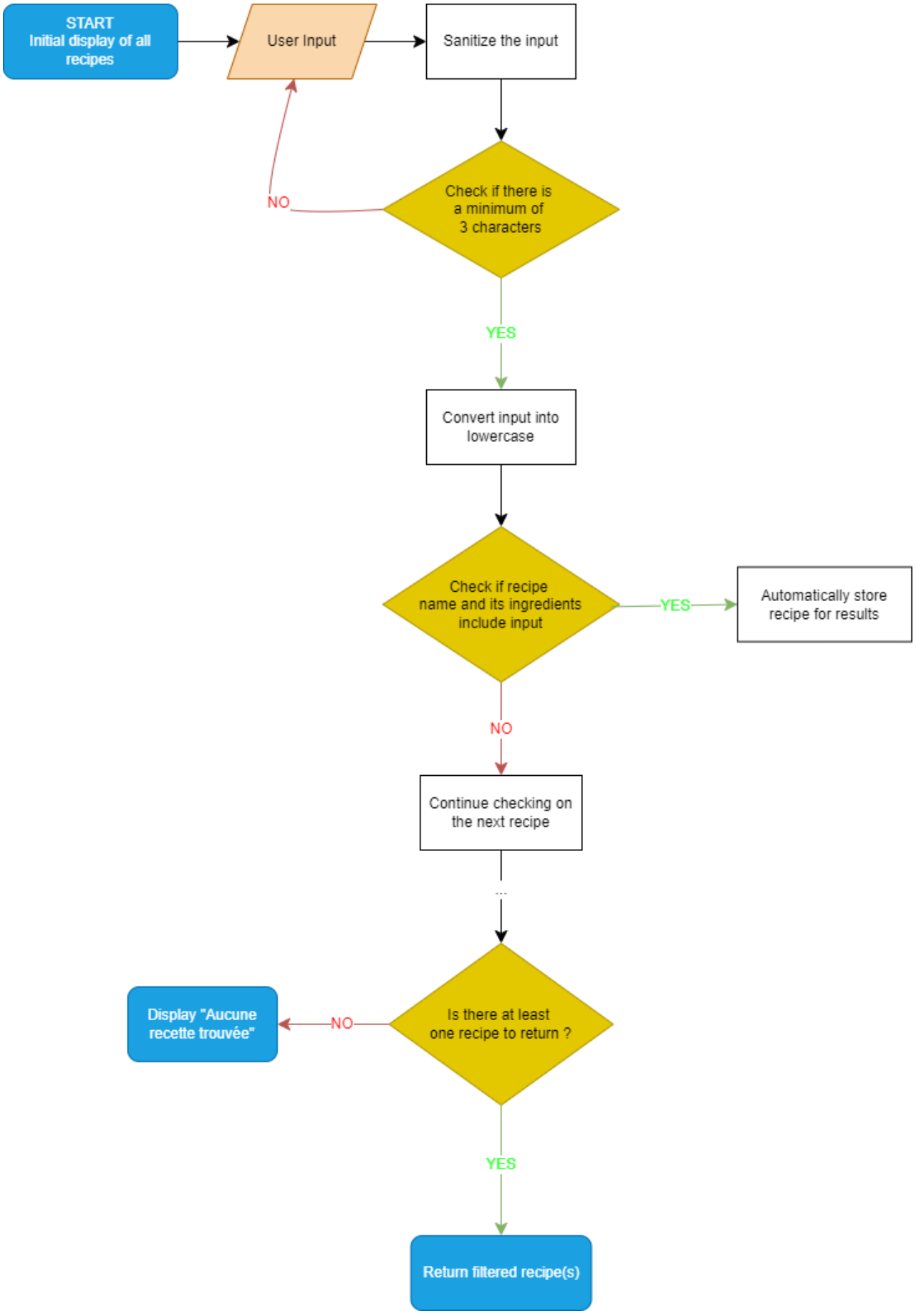


Figure 3 – Algorigramme – Version « Native Loops »