Fiche investigation de fonctionnalité

Fonctionnalité : Filtrer les recettes dans l'interface utilisateur	**Fonctionnalité #1**
Problématique : Accéder rapidement à une recette correspondant à un besoin de l'utilisateur dans les recettes déjà reçues.	

Option 1 : Utiliser des méthodes liées à l'objet tableau

Balayer la table de toutes les recettes en utilisant les méthodes « filter », et « some » afin de rechercher la séquence de caractères entrée dans le nom, ou la description, ou les ingrédients de la recette :
Si trouvé, on ajoute la recette au tableau de résultats

Veiller à l'ordre des tests pour :

- Réaliser les tests les plus rapides en tête (nom puis description puis ingrédients)
- Interrompre le test dès qu'en correspondance est trouvée (méthode « some » pour balayage de tableau, opérateur '||')

Avantages	**Inconvénients**
⊕ Utilisation d'un code dédié aux parcours de tables donc à priori plus performant ⊕ Code plus compact	⊖ Code moins 'intuitif' et moins 'naturel' ⊖

Nombre de caractères à remplir pour déclencher la recherche : 3
Nombre de caractères supplémentaires provoquant une nouvelle recherche : 1

Option 2 : Utiliser une boucle sur toutes les recettes

Balayer la table des recettes par une boucle principale suivie pour chaque recette :
Une boucle for…in balaie la liste de toutes les recettes. Pour chaque recette, rechercher dans le nom puis dans la description puis par une boucle for..in dans les ingrédients.

Veiller à l'ordre des tests pour :

- Réaliser les tests les plus rapides en tête (nom puis description puis ingrédients)
- Interrompre le test dès qu'en correspondance est trouvée (méthode « some » pour balayage de tableau, opérateur '||')

Avantages	**Inconvénients**
Code plus naturel Lisibilité meilleure	Code plus complexe à écrire et donc à maintenir Risque de bugs plus important (indices nombreux à gérer)

Nombre de caractères à remplir pour déclencher la recherche : 3
Nombre de caractères supplémentaires provoquant une nouvelle recherche : 1

Solution retenue :

Tests effectués avec JSBench.me & JSBen.ch

JSBENCHME				ALGO1 FILTER		ALGO2 FOR	
	JSBenchMe.FOU	16 resultats	Recherche "fou" sans tags	24948 ops	Fastest	12578 ops	49% slower
	JSBenchMe.PAI	5 resultats	Recherche "pai" sans tags	21052 ops	Fastest	9717 ops	54% slower
	JSBenchMe.CUI	28 resultats	Recherche "cui" sans tags	29512 ops	Fastest	14615 ops	50% slower
	JSBenchMe.PARACHUTE	0 Resultat	Recherche "parachute" sans tags	18332 ops	Fastest	10829 ops	41% slower
	JSBenchMe.BAL	1 résultat	Recherche "bal" sans tags	21585 ops	Fastest	10733 ops	44% slower
JSBEN.CH							
	JSBen.ch-FOU	16 resultats	Recherche "fou" sans tags	52471	Fastest	29920	57% slower
	JSBen.ch-PAI	5 resultats	Recherche "pai" sans tags	37541	Fastest	24553	65% slower
	JSBen.ch-CUI	28 resultats	Recherche "cui" sans tags	58294	Fastest	34120	58% slower
	JSBen.ch-PARACHUTE	0 Resultat	Recherche "parachute" sans tags	41150	Fastest	24758	60% slower
	JSBenchMe.BAL	1 résultat	Recherche "bal" sans tags	45436	Fastest	25686	56% slower

Conclusion : Choix Algo 1 (filter & some) plus performant pour différents profils de recherche

Annexes

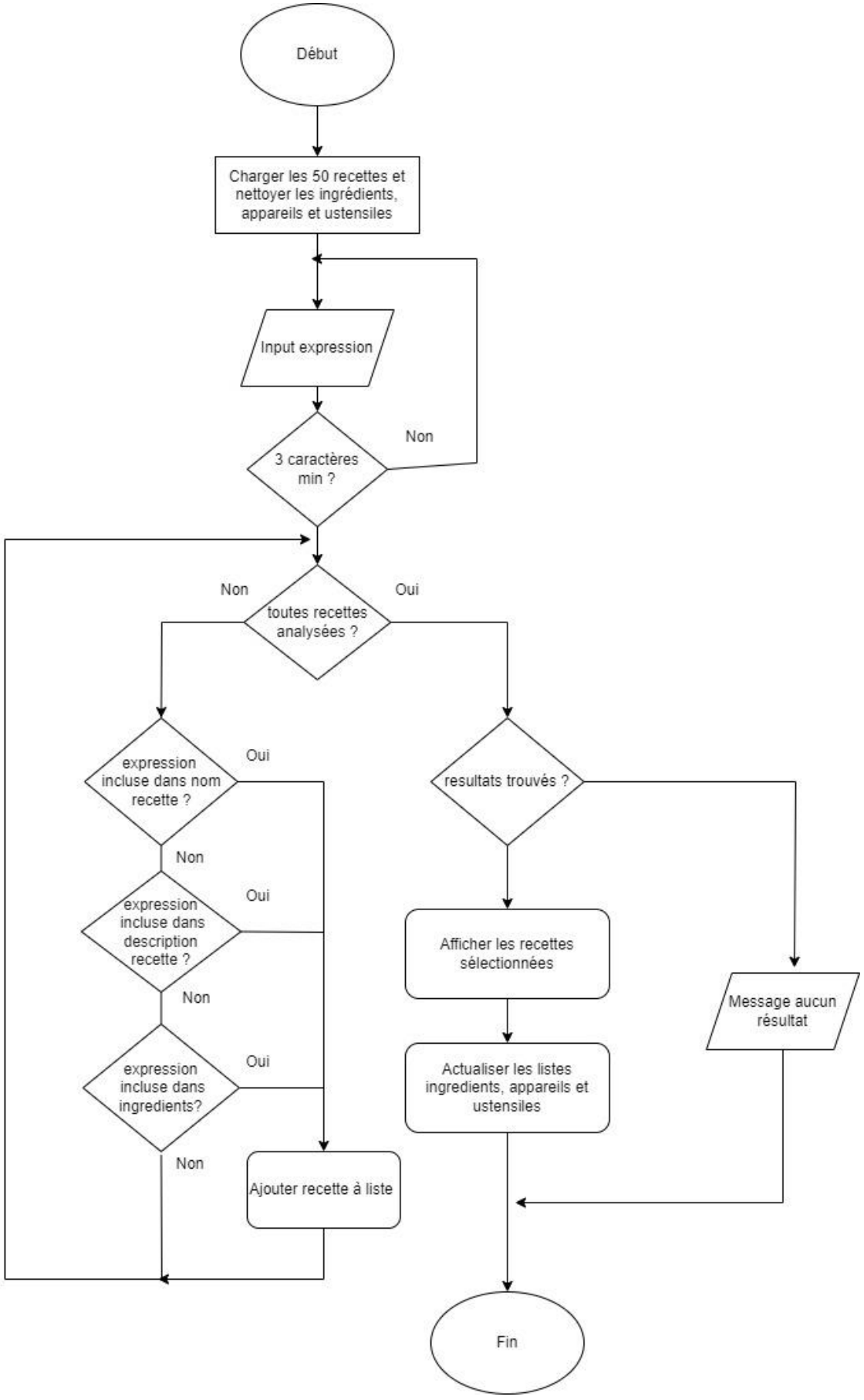

Figure 1 – Scenario nominal

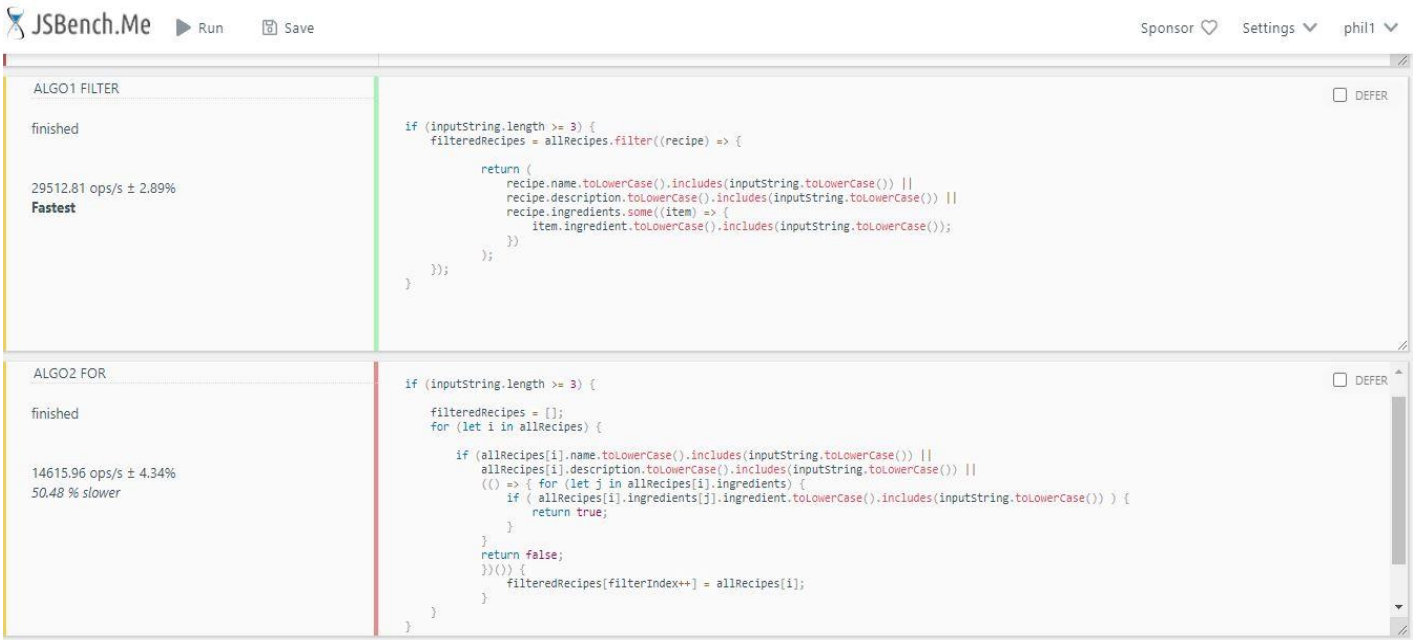

Figure 2 : TestBench.me