Fiche d'investigation de fonctionnalité

Fonctionnalité : Recherche de texte dans le champ principale	**Fonctionnalité #1**
Problématique : La fonctionnalité recherche est très importante. L'objectif est de tester 2 versions pour implémenter une recherche le plus efficace possible	

Option 1 : Méthode objet Array (foreach, filter, map,..) : Cette option consiste à s'appuyer sur les méthodes natives fournies par l'objet Array pour effectuer les opérations de recherche, filtrage et la transformation des données.

Avantages :

- **Performances optimisées** : les méthodes natives sont écrites en C++.
- **Lisibilité et maintenabilité** : code court, et compréhensif.
- **Moins d'erreurs potentiels** : manipulation de moins de logique, d'indice de boucle.
- **Approche fonctionnelle :** facile l'écriture d'un code déclaratif.

Inconvénients :

- **Moins de contrôle de bas niveau :** délégation au moteur JS.
- **Debug parfois moins granulaire :** une boucle permet de tracer plus facilement les étapes dans le code.
- **Moins flexible pour un algorithme très spécifique :** pour une recherche avancée et spécifiques, pas de degrés de liberté pour optimiser.

Option 2 : Méthode objet natives (for, while, …) : Cette option consiste à parcourir et comparer les données avec des boucles for, while, ..

Avantages :

- **Contrôle très fin de l'algorithme** : on peut décider exactement comment on parcourt les données, où on s'arrête, quelles optimisations spécifiques doivent être faites.
- **Possibilité d'algorithmes sur mesure** : utile si l'on veut implémenter une logique très particulière et performante pour une recherche.
- **Réduction des structures temporaires** : permet d'éviter les tableaux intermédiaires en remplissant directement le tableau résultat, ce qui réduit les allocations mémoire.

Inconvénients :

- **Moins performant dans notre cas d'usage** : les tests de performance j**sBench** montrent que cette option est environ 10× fois plus lente que l'option 1 (voir résultats page suivante).
- **Code plus long et moins lisible : m**ultiplication des boucles, des index, des conditions, des flags. Cela rend le code difficile à lire et à maintenir.
- **Risque accru de bugs : e**rreurs d'index, de conditions d'arrêt incorrectes, de non prise en compte des effets de bord.

Solution retenue :

L'option 2, basée sur les boucles natives, offre un contrôle très fin de l'algorithme mais au prix d'un code plus complexe et nettement moins performant.
Les mesures jsBench montrent qu'elle est environ dix fois plus lente que l'option 1, qui s'appuie sur les méthodes natives de l'objet Array. Pour notre fonctionnalité de recherche, l'option 1 est à privilégier. Par conséquent, l'option est retenue.

Fiche d’investigation de fonctionnalité

Tests de performance, résultats :

Conditions de test :

1. Données : base 35 recettes, texte recherché « coco »
2. Exécuter sous chrome
3. Outil : JSBEN.CH

Vous pouvez retrouver la configuration en ligne et exécuter le test en ligne ;

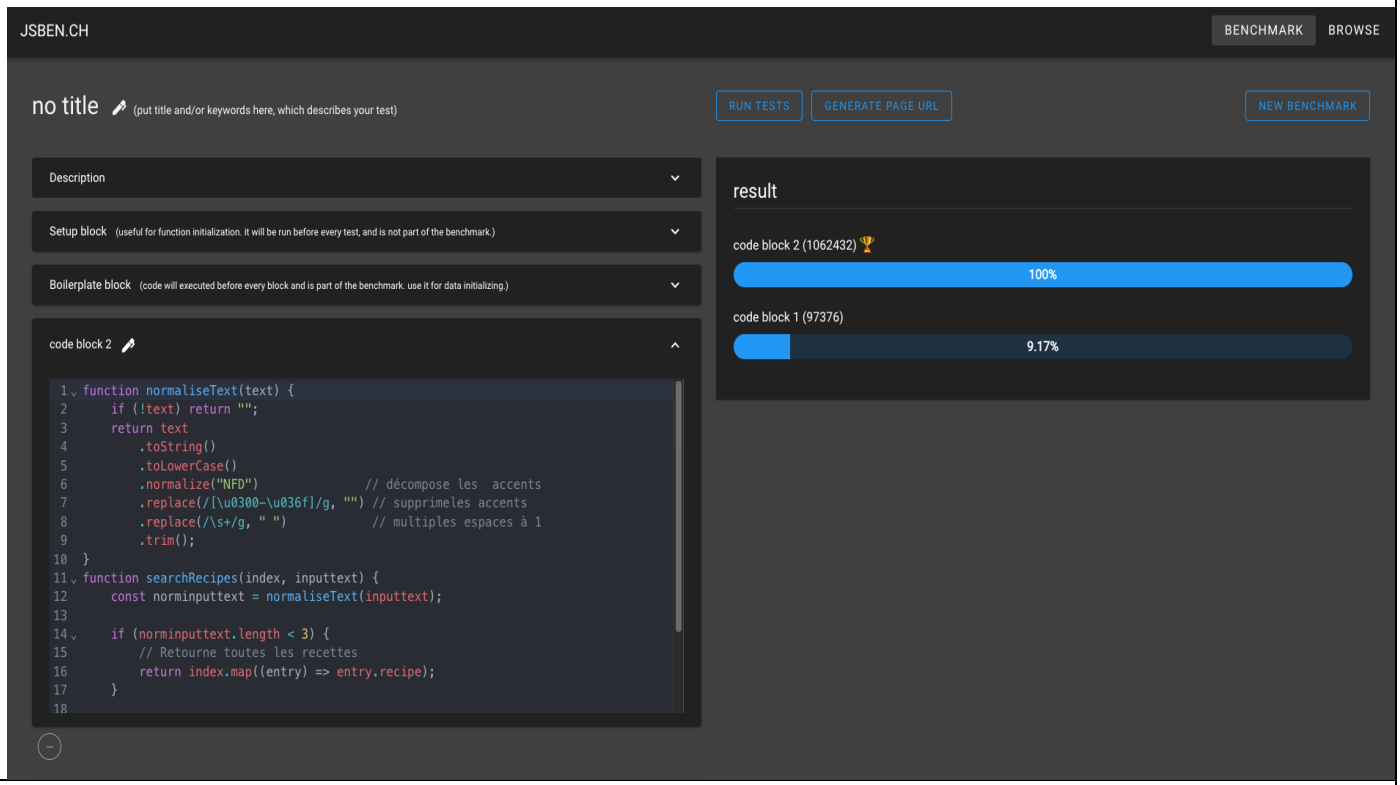
https://jsben.ch/JzzdP

Résultats affichés :

- code block 2 (option 1) : 1 062 432 opérations/sec → 100 %
- code block 1 (option 2): 97 376 ops/sec → 9,17 %

Conclusion :

- **code block 2 (option 1) est environ 10,9 fois plus rapide que code block 1.**
  (1 062 432 ÷ 97 376 ≈ 10,9)



Fiche d’investigation de fonctionnalité