



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 : <ul style="list-style-type: none"> 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 : <ul style="list-style-type: none"> 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 : <ul style="list-style-type: none"> 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 remplaçant directement le tableau résultat, ce qui réduit les allocations mémoire. | Inconvénients : <ul style="list-style-type: none"> Moins performant dans notre cas d'usage : les tests de performance jsBench montrent que cette option est environ 10x fois plus lente que l'option 1 (voir résultats page suivante). Code plus long et moins lisible : multiplication des boucles, des index, des conditions, des flags. Cela rend le code difficile à lire et à maintenir. Risque accru de bugs : erreurs 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.**
 $(1\ 062\ 432 \div 97\ 376 \approx 10,9)$

The screenshot shows the JSBEN.CH web application interface. At the top, there are buttons for 'BENCHMARK' and 'BROWSE'. Below that, a title 'no title' is followed by a note '(put title and/or keywords here, which describes your test)'. There are three main sections: 'Description', 'Setup block', and 'Boilerplate block'. The 'code block 2' section contains the following JavaScript code:

```
1< function normaliseText(text) {  
2   if (!text) return "";  
3   return text  
4     .toString()  
5     .toLowerCase()  
6     .normalize("NFD")           // décompose les accents  
7     .replace(/[\u0300-\u036f]/g, "") // supprime les accents  
8     .replace(/\s+/g, " ")        // multiples espaces à 1  
9     .trim();  
10 }  
11< function searchRecipes(index, inputtext) {  
12   const norminputtext = normaliseText(inputtext);  
13  
14   if (norminputtext.length < 3) {  
15     // Retourne toutes les recettes  
16     return index.map((entry) => entry.recipe);  
17   }  
18 }
```

To the right, under the 'result' section, there is a chart comparing the performance of 'code block 2 (1062432)' and 'code block 1 (97376)'. The chart shows a blue progress bar for each, with 'code block 2' at 100% and 'code block 1' at 9.17%.



Fiche d'investigation de fonctionnalité

