


Fiche d'investigation de fonctionnalité

| | |
|---|--|
| Fonctionnalité : Recherche principal | Fonctionnalité#1 |
| <p>Problématique : Filtrer les recettes selon les besoins de l'utilisateur parmi celles déjà reçues.</p> <p>La barre principale permet de rechercher des mots ou groupes de lettres dans le titre, les ingrédients ou la description.</p> | |
| <p>Option 1 : Native</p> <p>Dans cette approche, nous utilisons des boucles itératives natives telles que for pour manipuler les tableaux.</p> | |
| <p>Avantages</p> <ul style="list-style-type: none"> ★ Utilisation des connaissances de base pour créer un algorithme de recherche. | <p>Inconvénients</p> <ul style="list-style-type: none"> ★ Réinvention de la roue, reproduisant quelque chose qui est déjà implémenté. ★ Code plus long |
| <p>Nombre de champs : 1 champ de recherche Nombre de champs minimum à remplir : 0 (afficher toutes les recettes)</p> | |
| <p>Option 2 : Fonctionnelle</p> <p>Dans cette approche, nous utilisons des méthodes avancées pour manipuler les tableaux.</p> | |
| <p>Avantages</p> <ul style="list-style-type: none"> ★ Ecrire du code plus lisible. ★ Efficacité et gain en productivité. | <p>Inconvénients</p> <ul style="list-style-type: none"> ★ Nécessite une connaissance des méthodes de manipulation des tableaux (filter, some, sort...) en JavaScript et une compréhension de leur fonctionnement |
| <p>Nombre de champs: 1 champ de recherche Nombre de champs minimum à remplir : 0 (afficher toutes les recettes)</p> | |
| <p>Solution retenue :</p> <p>En se basant sur les résultats obtenus par les outils de comparaison de performance, les deux méthodes se sont avérées performantes. Cependant, pour des raisons d'efficacité et de productivité, la deuxième option, utilisant les méthodes de l'objet array (Fonctionnelle), a été retenue.</p> | |

Mesure de performance des deux méthodes

Pour mesurer les performances d'un code js, on peut faire le test manuel avec du code en utilisant `console.time` et `console.timeEnd`.

Terme recherchée : banane

 127.0.0.1:5501/?search=banane

| Calcul de temps d'exécution en millisecondes "ms" | | |
|---|---|---|
| | Native | Fonctionnelle |
| Essai 1 | Algorithme 1 : 73 ms - chronomètre arrêté | Algorithme 2 : 53 ms - chronomètre arrêté |
| Essai 2 | Algorithme 1 : 57 ms - chronomètre arrêté | Algorithme 2 : 62 ms - chronomètre arrêté |
| Essai 3 | Algorithme 1 : 75 ms - chronomètre arrêté | Algorithme 2 : 62 ms - chronomètre arrêté |

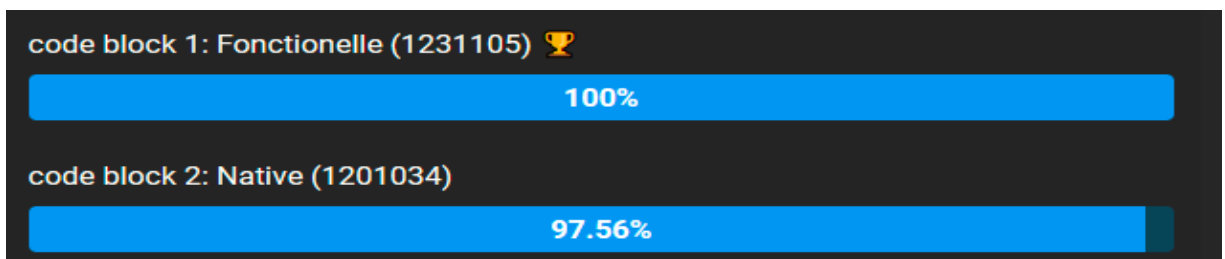
Pour la méthode "native" : Ces valeurs indiquent que l'exécution du code a pris respectivement 73, 57 et 75 millisecondes.

Pour la méthode "Fonctionnelle" : A pris respectivement 53, 62 et 62 millisecondes.

On observe que les tendances générales dans le temps d'exécution sont inférieures à celles de la méthode "fonctionnelle". Cela pourrait indiquer que la méthode "fonctionnelle" est plus efficace. Cependant, il est important de prendre en compte d'autres facteurs tels que la lisibilité du code et sa maintenabilité.

Test de performance avec jsben.ch

On peut aussi utiliser des outils de comparaison de performance tels que : ★jsBen.ch



Les résultats du test montrent que les deux approches ont démontré des performances globalement satisfaisantes. Cependant, en considérant les résultats, ainsi que mes propres préférences et exigences spécifiques pour le projet, j'ai choisi d'opter pour la méthode fonctionnelle. Bien que la différence de 2,44% entre les performances des deux approches puisse être considérée comme relativement faible, cette décision est basée sur ma conviction que la méthode fonctionnelle répondra le mieux à mes besoins en termes de lisibilité, efficacité et maintenabilité du code

Diagramme de la recherche avancée

