

Fiche d’investigation de fonctionnalité

|  |
| --- |
| **Fonctionnalité :** Recherche principale |
| **Problématique:** Afin d’offrir aux utilisateurs une recherche la plus rapide possible, nous allons en ecrire deux versions. Dans un second temps, nous allons utiliser une application web Jsben.ch, qui va éprouver ces deux versions l’une envers l’autre et sélectionner la plus rapide. |

|  |
| --- |
| **Option 1 :** Utiliser la fonction native filter() |
| **Avantage:**  Sur le banc d’essai de Jsben.ch, on peut constater que cet algorithme est plus rapide, cela est du aux function native, mais aussi grace au chargement plus court, car à chaque keyup, on fait la recherché sur les données de la précedente recherché.  Il y a une lecture simple du code. |
| **Inconvénient:**  Cet version par ça fonction native est la plus rapide, or ce n’est pas le cas. |

|  |
| --- |
| **Option 2 :** Utiliser une boucle for() |
| **Avantage:**  La lecture approfondit du est plus proche de la réalité, car on ne voit pas ce qu’il y a derrière les function native. |
| **Inconvénient:**  Lisibilité du code un peu moin claire.  Plus de calcul, se qui le rend moin performant. |

|  |
| --- |
| **Solution retenue :** |
| Sur 50 recettes, la différence entre les deux options ne se voit pas en termes d’ergonomie pour l’utilisateur.  Par contre, pour savoir ce qu’il va se passer lorsque cette base de données va évoluer, il est bon de savoir qu’elle fonction est la plus rapide, grâce à des outils de comparaison.  Les concepteurs du langage Javascript cherche à ce que leurs fonctions natives soient les  plus performante possible, mais leur utilization peut être plus ou moin bien utilisé.  C’est pour cela que des outils de comparaison de code tel que **jsben** exsistent.  Ci-joint le lien du benchmark : <https://jsben.ch/XSslk> |



