# Fiche d'investigation de fonctionnalité
# Filtrage dynamique des recettes

**Fonctionnalité :**
Filtrage par méthode forEach et .ma

**Problèmatique :**
Afin de retenir un maximum d'utilisateur, nous recherchons la vitesse de chargement et d'affichage la plus rapide et fluide possible sachant que les utilisateurs peuvent les recettes selon 2 axes : le 1er par une recherche dans les titres, les ingrédients et les descriptions des recettes, le 2ème selon trois filtres avancés qui se concentre sur les ingrédients, les appareils et les ustensiles. Les 2 axes de recherche peuvent également se faire conjointement. Pour ce faire nous comparons 2 méthodes. De plus, nous avons choisit un code orienté objet pour la base du code.

**Option 1 : Utilisation de la méthode forEach**
Dans cette option, nous utilisons la méthode forEach pour filtrer et afficher les éléments que souhaiter par l'utilisateur par la saisie dans les inputs (principal et des filtres avancés) et / ou le choix de mots clé dans les listes de filtres avancés

**Avantages :**
- Simpla à utliser : forEach() est facile à comprendre et à utiliser. Il suffit de fournir une fonction de rappel qui sera exécutée pour chaque élément du tableau.
- Modifie directement le tableau : Vous pouvez modifier directement le tableau en utilisant cette méthode, car il parcourt simplement chaque élément du tableau et exécute une action pour chacun d'eux.
- Souvent plus optimisé pour les opérations simples, car il parcourt chaque élément du tableau et exécute une action pour chacun d'eux
- N'impliquant pas la création d'un nouveau tableau, il n'y a donc pas de surcharge de mémoire liée à la création de nouveau tableau.

**Inconvénients :**
- Ne retourne pas de nouveau tableau : cette méthode ne retourne rien, du coup les résultats ne sont pas stockés dans de nouvelles variables directement. Cela peut rendre le code un peu moins lisible pour l'enchainement de plusieurs opérations.
- Moins adapté pour les transforamtions de tableau : si on a besoin de transformer chaque élément du tableau et de récupérer un nouveau tableau avec ces transformations, cette méthode n'est pas l'outil le plus approprié, car il ne retourne pas de nouveau tableau
- Chaque itération est effectuée séquentiellement. Cela peut entrainer un temps d'exécution plus longs lorsque l'on à un grand volume de données à traiter.
-

**Option : Utilisation de la méthode .map**
La méthode .map() parcourt chaque élément d'un tableau et retourne un nouveau tableau contenant les résultats de l'application d'une fonction à chaque élément. C'est utile lorsque l'on souhaite transformer chaque élément du tableau en quelque chose d'autre et obtenir un nouveau tableau avec ces transformations.

| **Avantages :** | **Inconvénients :** |
|---|---|
| - La méthode .map() retourne un nouveau tableau contenant les résultats des transformations appliqués à chaque élément du tableau d'origine. Cela rend le code plus clair et plus lisible.<br>- Les transformations sur chaque élément du tableau peuvent être exécutées simultanément. Cela peut entrainer un gains de vitesse significatifs lorsque l'on à un grand volume de données à traiter. | - Cette méthode crée un nouveau tableau systématiquement. Cela peut être un inconvénient lorsque l'on souhaite modifier directement le tableau d'origine.<br>- Cette méthode est légèrement plus complexe que forEach quand on débute dans le code en raison de sa syntaxe<br>- Cette méthode crée un nouveau tableau contenant les résultats des transformation appliquées à chaque élément du tableau d'origine. La création de ces nouveaux tableaux peut entrainer une surcharge de mémoire et une légère baisse de performance lorsque l'on travaille sur de très grands ensembles de données |

En résumé : Pour les opérations simples et lorsque la création d'un nouveau tableau n'est pas nécessaire, forEach peut être plus rapide. Cependant, lorsque l'on a besoin de transformations complexes sur chaque élément, .map peut offrir des performance supérieures.