

Fiche d'investigation de fonctionnalité

Filtrage dynamique des recettes

Fonctionnalité :

Filtrage par méthode `forEach`, `.map` et `for`

Problématique :

Afin de retenir un maximum d'utilisateurs, nous recherchons la vitesse de chargement et d'affichage la plus rapide et fluide possible sachant que les utilisateurs peuvent chercher les recettes selon 2 axes : le 1^{er} par une recherche dans les titres, les ingrédients et les descriptions des recettes, le 2^{ème} 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 3 méthodes. De plus, nous avons choisi un code orienté objet pour la base du code.

Option 1 : Utilisation de la méthode `forEach`

```
array.forEach(function(element) {});
```

Dans cette option, nous demandons à Java Script de parcourir le tableau spécifié et va appeler et exécuter les fonctions fournies pour chaque élément.

Donc chaque élément est soumis à la liste des fonctions jusqu'à la fin du tableau

Avantages :

- La syntaxe est plus simple et plus claire que celle d'une boucle `for`.
- La syntaxe peut faciliter la lecture et la compréhension du code
- Cette méthode est idéale lorsque l'on veut simplement parcourir tous les éléments d'un tableau et effectuer une action pour chacun d'eux.
- Souvent plus optimisé par rapport à `.map` 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'enchaînement de plusieurs opérations.
- Moins adapté pour les transformations 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 entraîner un temps d'exécution plus longs lorsque l'on a un grand volume de données à traiter.

Option : Utilisation de la méthode .map

```
const newArray = array.map(function(element) {  
  return  
});
```

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 :

- 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.
- Les transformations sur chaque élément du tableau peuvent être exécutées simultanément. Cela peut entraîner un gain de vitesse significatifs lorsque l'on a un grand volume de données à traiter.

Inconvénients :

- Cette méthode crée un nouveau tableau systématiquement. Cela peut être un inconvénient lorsque l'on souhaite modifier directement le tableau d'origine.
- Cette méthode est légèrement plus complexe que forEach quand on débute dans le code en raison de sa syntaxe
- Cette méthode crée un nouveau tableau contenant les résultats des transformations appliquées à chaque élément du tableau d'origine. La création de ces nouveaux tableaux peut entraîner une surcharge de mémoire et une légère baisse de performance lorsque l'on travaille sur de très grands ensembles de données

Option : Utilisation de la méthode for

```
for (let i = 0; i < array.length; i++) {}
```

Avec la méthode for, on a un contrôle total sur le processus de parcours des données. On spécifie explicitement comment parcourir le tableau et ce que l'on veut faire avec chaque élément

Avantages :

- On a un contrôle complet sur le processus, y compris la possibilité de l'arrêter en cours avec 'break', ce qui est impossible avec forEach et .map.
- For ne traite les données que selon les conditions fournis en paramètre.
- Permet la recherche d'élément spécifique pour les traiter.

Inconvénients :

- La syntaxe est plus longue et plus difficile à lire.
- Peut-être plus sujet à erreur (boucle infini, erreur d'indexation...) si elle n'est pas correctement écrite.

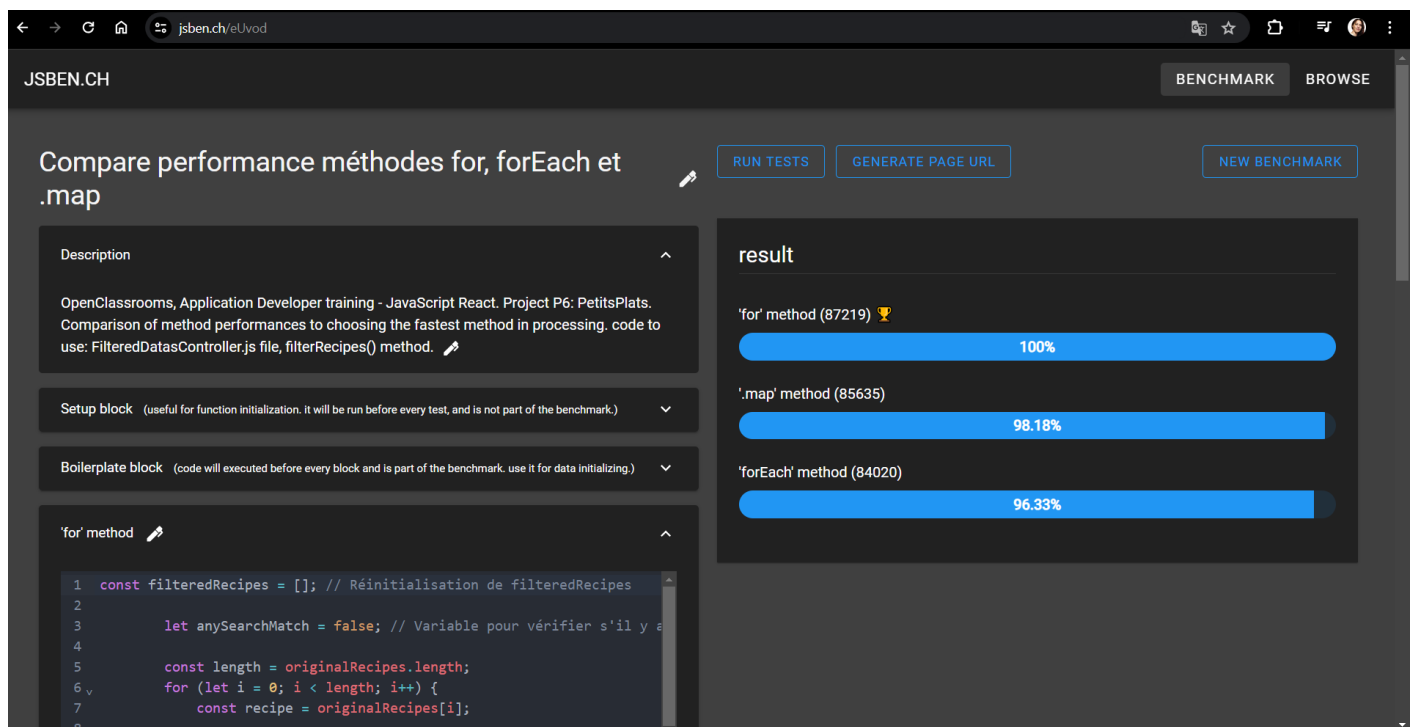
En résumé : `forEach` est principalement utile pour parcourir un tableau et effectuer une action sur chaque élément. Il pourra être plus efficace sur des opérations simples.

`.map` est utile pour transformer les données d'un tableau. Son traitement simultané des données peut le rendre plus rapide sur le traitement de grand ensemble de données par rapport à `forEach`, mais peut rendre le calcul du traitement plus lourd.

`For` offre un contrôle parfait sur la manière dont sont parcourus les données et le traitement des données, cela le rend plus performant que `forEach` et `.map`, mais le rend plus complexe dans la syntaxe et la lecture.

L'idéale serait d'utiliser ces méthodes en fonction des besoins dans l'ensemble des fichiers.

Analyse JSBen (<https://jsben.ch/a14lw>) :



Conclusion :

Après plusieurs analyses sur le site jsben.ch, nous retiendrons la méthode 'for'.

(Sur 10 tests réalisés, 'for' réussi 4/10, 'forEach' réussi 3/10 et '.map' réussi 3/10).

Il faudra tout de même retenir que selon ce prototype, les écarts entre les méthodes varient très peu (moins de 4% d'écart à chaque test) et que la meilleure performance ne revient pas toujours à la méthode 'for'. Il faudra donc préconiser d'utiliser un mélange de ces méthodes en fonction des besoins.

Algorithme du site (algorithme identique à toutes les méthodes : for, forEach, .map)

