

Fiche d'investigation de fonctionnalité

Fonctionnalité : Filtrage de recettes

Problématique : Afin de permettre à l'utilisateur de filtrer rapidement les recettes à l'aide de critères, nous devons comparer les avantages et inconvénients de l'utilisation de la boucle `for` et de la méthode `filter`.

Option 1 : Utilisation de la boucle `for`

Dans cette option, nous utilisons une boucle `for` classique pour parcourir toutes les recettes et appliquer les filtres manuellement. Chaque recette est vérifiée par rapport aux critères de recherche et ajoutée au tableau de résultats si elle correspond.

Avantages

- ⊕ **Contrôle total sur la logique** : La boucle `for` permet de personnaliser facilement la logique de filtrage pour chaque critère.
- ⊕ **Performances** : Dans certains cas, la boucle `for` peut être plus rapide, car elle permet de sortir de la boucle dès qu'une recette est rejetée, sans continuer à évaluer d'autres critères.
- ⊕ **Flexible** : Permet d'ajouter des conditions supplémentaires ou des traitements spécifiques sans trop de complexité.

Inconvénients

- ⊖ **Moins lisible** : Le code peut devenir difficile à maintenir si le filtrage devient complexe, car il faut ajouter plusieurs conditions dans la boucle.
- ⊖ **Plus de lignes de code** : Cela nécessite de gérer manuellement la logique de filtrage et de créer des tableaux temporaires.
- ⊖ **Code redondant** : Chaque critère de filtrage doit être écrit explicitement, ce qui peut entraîner de la répétition dans le code.

Option 2 : Utilisation de la méthode `filter`

La méthode `filter` est utilisée pour créer un nouveau tableau contenant les recettes qui correspondent aux critères de recherche. Cette approche est généralement plus concise et s'appuie sur la capacité de `filter` à appliquer une fonction de test sur chaque élément.

Avantages

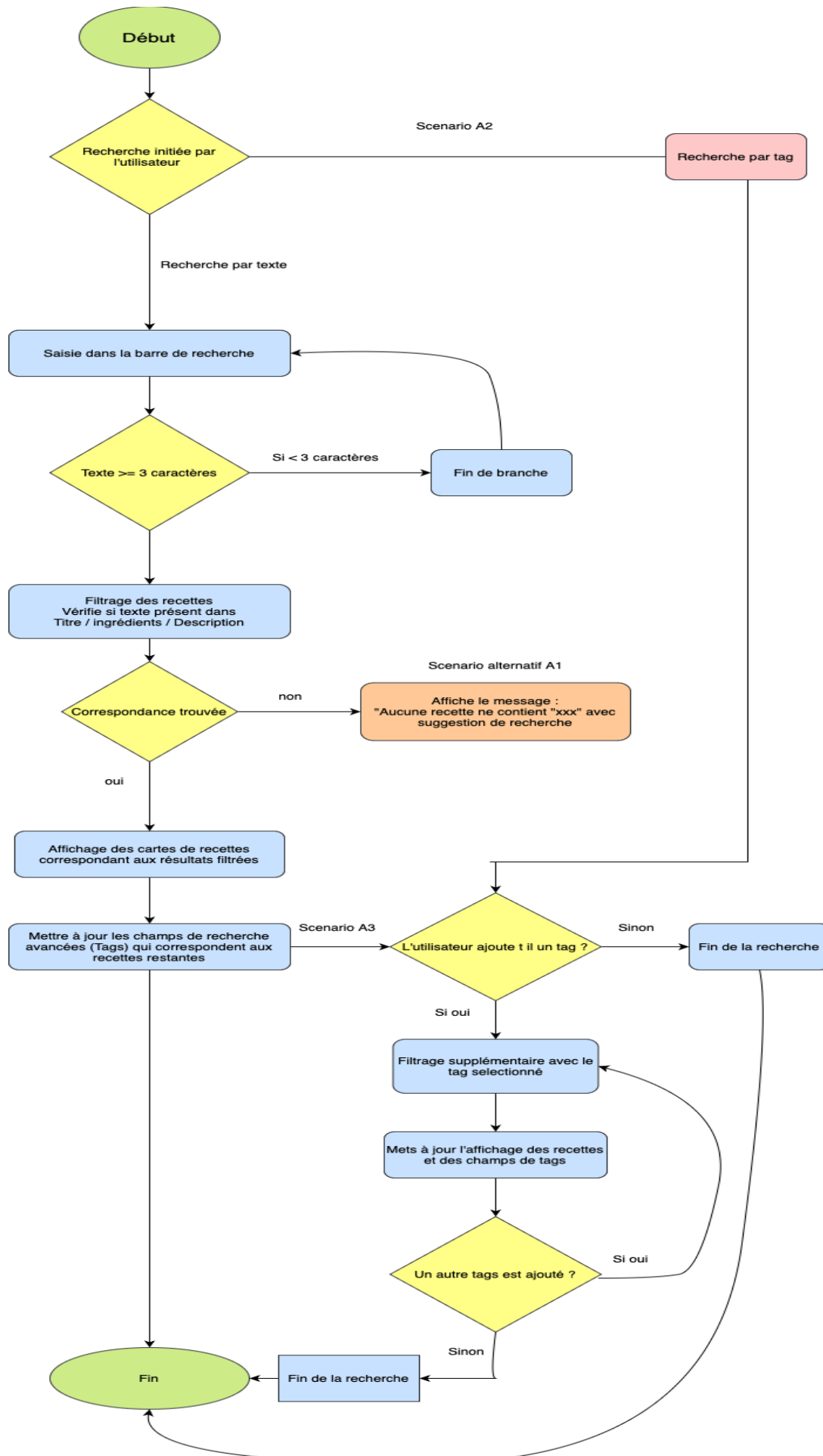
- ⊕ **Code plus lisible** : La méthode `filter` rend le code plus concis et facile à comprendre, surtout lorsqu'il s'agit de filtrer en fonction de plusieurs critères simples.
- ⊕ **Facilité de maintenance** : Si de nouveaux critères sont ajoutés, la méthode `filter` reste claire et facile à étendre.
- ⊕ **Moins de code** : Pas besoin de gérer manuellement l'ajout des recettes dans un tableau, car `filter` le fait automatiquement.
- ⊕ **Fonctionnement immuable** : `filter` crée un nouveau tableau sans modifier l'original, ce qui peut être utile pour éviter les effets secondaires.

Inconvénients

- ⊖ **Moins flexible** : La méthode `filter` est plus limitée si tu as des besoins complexes en matière de filtrage, car elle ne permet pas d'intégrer facilement des logiques très personnalisées (comme des calculs spécifiques dans chaque itération).
- ⊖ **Performance** : Si les critères sont complexes ou s'il y a un grand nombre d'éléments, `filter` peut être légèrement moins performant que la boucle `for`, car il parcourt tout le tableau sans possibilité de sortir prématurément.

Solution retenue

J'ai retenu l'utilisation de la méthode `filter` pour son **code plus lisible** et sa **maintenabilité**. Elle nous permet de gérer facilement le filtrage sur plusieurs critères (comme le texte de recherche et les tags) et de le rendre évolutif si de nouveaux critères de filtrage sont ajoutés par la suite. Bien que la boucle `for` puisse offrir plus de flexibilité et des performances potentielles pour des logiques complexes, `filter` reste suffisant pour les cas d'usage actuels et améliore la clarté du code.




Bloc numéro 1 : Code block boucle for ()


Bloc numéro 2 : Code block filter ()


JSBEN.CH


BENCHMARKBROWSE


no title  (put title and/or keywords here, which describes your test)


RUN TESTSGENERATE PAGE URLNEW BENCHMARK

Description 

Site plats 


Setup block (useful for function initialization. it will be run before every test, and is not part of the benchmark.) 

Boilerplate block (code will executed before every block and is part of the benchmark. use it for data initializing.) 

code block 1 

```
1 // Mise à jour du compteur de recettes
2 function updateRecipeCount(count) {
3   nbRecipeSpan.textContent = `${count} recette${count > 1 ?
4 }
5
6 // Filtre les recettes en fonction du texte de recherche (ut
7 function filterRecipesByText(searchText) {
8   const lowerSearchText = searchText.toLowerCase();
9   const result = [];
10  for (let i = 0; i < recipes.length; i++) {
11    const recipe = recipes[i];
12    if (
13      recipe.name.toLowerCase().includes(lowerSearchText) ||
```

result

code block 1 (12374160) 
100%

code block 2 (12278545)
99.23%