Fiche d'investigation de fonctionnalité

Marie Moore
3 Mars 2022
Les Petits Plats

Fonctionnalité: Recherche de recette
Problématique: Effectuer une recherche de recettes très rapide selon les ingrédients, ustensiles et description et afficher les résultats de manière simultanée.

Option 1: Programmation Déclarative avec forEach()

La méthode ForEach() permet d'exécuter une fonction callback sur chaque élément d'un tableau dans l'ordre croissant de l'indice. ForEach ne modifie pas le tableau sur lequel elle est appelée.

Avantages	Inconvenients
Pratique, car elle est disponible par défaut. Concise, car elle ne prend qu'une ligne de code.	L'appel d'une fonction callback sur chaque élément ralentit la performance sur un tableau de taille conséquente.

Performance:

forEach

finished

1426287.88 ops/s ± 0.62%
11.39 % slower

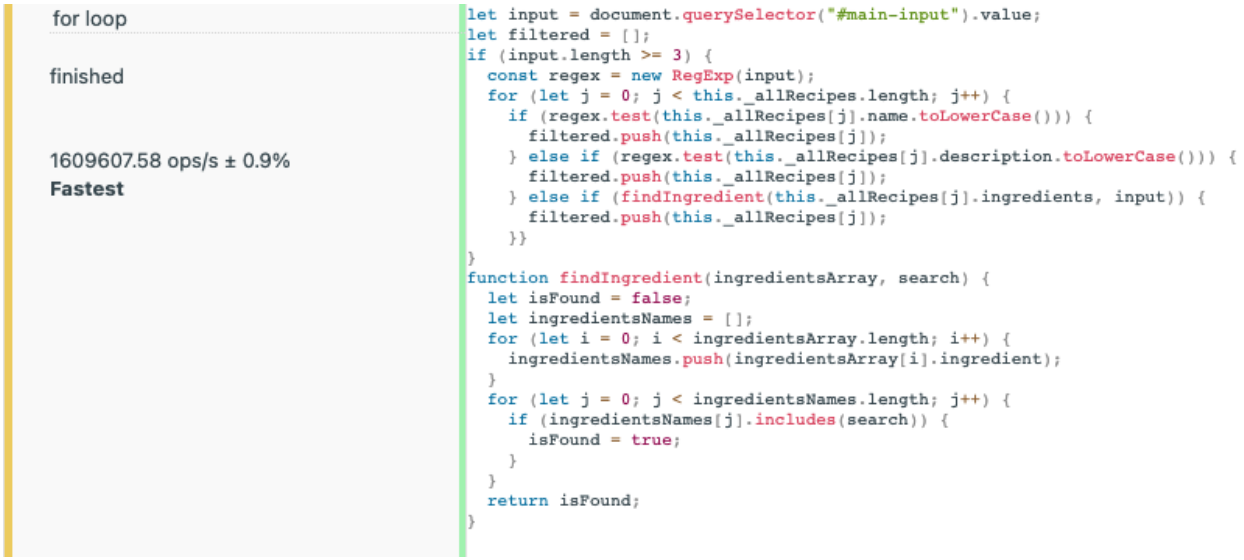
```
let input = document.querySelector("#main-input").value;
let filtered = [];
if (input.length >= 3) {
  const regex = new RegExp(input);
  this._allRecipes.forEach((element) => {
      if (regex.test(element.name.toLowerCase())) {
        filtered.push(element);
      } else if (regex.test(element.description.toLowerCase())) {
        filtered.push(element);
      } else if (findIngredient(element.ingredients, input)) {
        filtered.push(element);
      }
})}
function findIngredient(ingredientsArray, search){
    let isFound = false;
    let ingredientsNames = ingredientsArray.map(r => r.ingredient);
    for (let ingredient of ingredientsNames) {
      if (ingredient.includes(search)) {
        isFound = true;
      }}
    return isFound;
  }
```

Option 2: Programmation Impérative avec la boucle For

Une boucle for répète des instructions jusqu'à ce qu'une condition donnée ne soit plus vérifiée:
for ([expressionInitiale]; [condition]; [expressionIncrément])
instruction

Avantages	Inconvenients
Rapidité et performance Possibilité d'ajouter une fonction callback si besoin.	Le code de la programmation impérative est plus long à écrire et plus flou à comprendre: tout d'abord à cause de son nombre de lignes de code et aussi à cause de la présence d'index et de l' incrémentation.

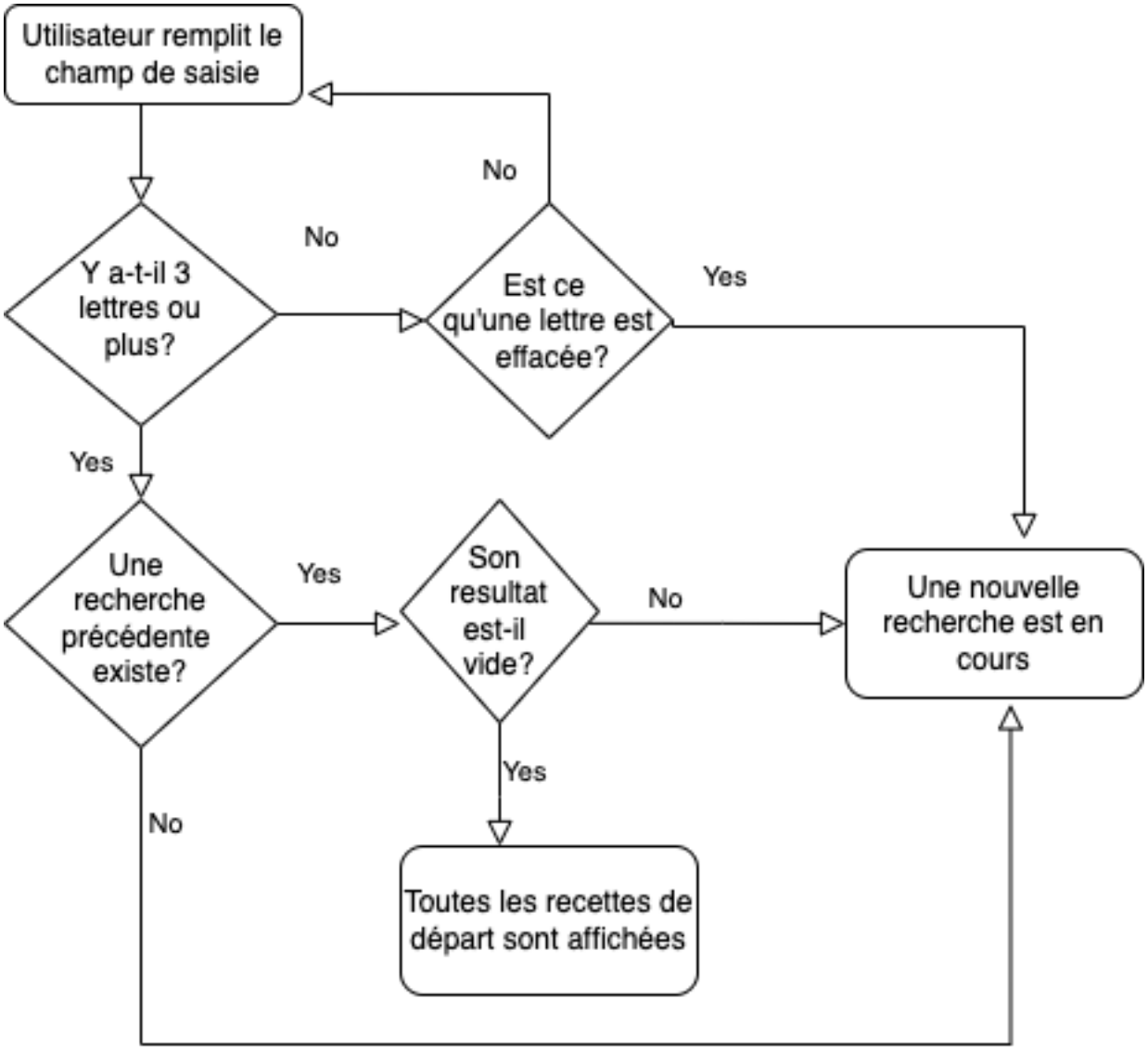
Performance:




```
let input = document.querySelector("#main-input").value;
let filtered = [];
if (input.length >= 3) {
  const regex = new RegExp(input);
  for (let j = 0; j < this._allRecipes.length; j++) {
    if (regex.test(this._allRecipes[j].name.toLowerCase())) {
      filtered.push(this._allRecipes[j]);
    } else if (regex.test(this._allRecipes[j].description.toLowerCase())) {
      filtered.push(this._allRecipes[j]);
    } else if (findIngredient(this._allRecipes[j].ingredients, input)) {
      filtered.push(this._allRecipes[j]);
    }}
}
function findIngredient(ingredientsArray, search) {
  let isFound = false;
  let ingredientsNames = [];
  for (let i = 0; i < ingredientsArray.length; i++) {
    ingredientsNames.push(ingredientsArray[i].ingredient);
  }
  for (let j = 0; j < ingredientsNames.length; j++) {
    if (ingredientsNames[j].includes(search)) {
      isFound = true;
    }
  }
  return isFound;
}
```

Solution Retenue

Après le test sur JSBENCH et la comparaison des résultats, je retiens l'option 2 qui utilise la programmation impérative avec la boucle for. Malgré sa lisibilité inférieure, la problématique principale du projet reste une performance optimale.


Utilisateur remplit le champ de saisie
No
No
Y a-t-il 3 lettres ou plus?
Est ce qu'une lettre est effacée?
Yes
Yes
Une recherche précédente existe?
Yes
Son resultat est-il vide?
No
Une nouvelle recherche est en cours
Yes
No
Toutes les recettes de départ sont affichées