Fiche d'investigation de focntionnalité

Fonctionalité : Moteur de recherche de recettes	Fonctionnalité #1
Problématique : Fournir une expérience utilisateur fluide pour rechercher des recettes par mots-clés et filtres dynamiques.	

Option 1 : Méthodes fonctionnelles (array-method) Dans cette option, nous utilisons les méthodes natives de l'objet Array, telles que .filter(), .map(), .some(), .includes(). Ces méthodes permettent de chaîner les traitements et d'écrire du code plus lisible et expressif. Elles sont particulièrement adaptées pour manipuler des tableaux comme la liste des recettes.	
Avantages ⊕ Syntaxe concise, lisible et moderne ⊕ Code facilement maintenable et réutilisable ⊕ Optimisé pour des interactions courantes avec les tableaux ⊕ Adapté aux manipulations DOM (génération HTML avec .map())	Inconvénients ⊖ Moins de contrôle étape par étape ⊖ Difficile à déboguer pas à pas pour un débutant ⊖ Moins performant dans certains cas extrêmes (ex : traitements lourds sur 100 000 éléments)

Option 2 : Méthodes fonctionnelles (loop-method) Dans cette option, nous utilisons des boucles classiques, comme for, for…of, et while, ainsi que des instructions if, break, et push() pour construire les résultats. Ce choix permet une meilleure maîtrise de chaque étape du traitement et une plus grande flexibilité dans les cas personnalisés.	
Avantages ⊕ Permet une gestion très fine des étapes ⊕ Plus pédagogique et plus facile à suivre pour un développeur débutant ⊕ Meilleur pour optimiser manuellement certaines portions de code	Inconvénients ⊖ Syntaxe plus longue et plus verbeuse ⊖ Moins lisible sur des traitements complexes imbriqués ⊖ Risque de duplication de logique (boucles imbriquées fréquentes)

Solution retenue :
La version array-method a été retenue pour **sa lisibilité, sa modularité et sa concision**, permet une mise en oeuvre rapide et maintenable du moteur de recherche, tout en répondant aux besoins du projet.

La version loop-method est utile pour former de nouveaux développeurs ou pour des cas où un **contrôle très fin des performances** est requis.

JSBench.Me ▶ Run Please login/register to save & publish tests

Sponsor ♡ Settings ∨ Sign in ∨

Les Petits Plats v1 - by .

enter test suite description

Setup HTML - click to add setup HTML

Setup JavaScript

```
const recipes = [
  {
    "id": 1,
    "image": "Recette01.jpg",
    "name": "Limonade de Coco",
    "servings": 1,
    "ingredients": [
      {
        "ingredient": "Lait de coco",
        "quantity": 400,
        "unit": "ml"
      }
```

Array-method

DEFER

```
const result = recipes.filter(recipe =>
 recipe.ingredients.some(ing =>
  ing.ingredient.toLowerCase().includes(query)
 )
);
```

finished

161 k ops/s ± 0.99%

18.38 % slower

Loop-method

DEFER

```
const result = [];

for (let i = 0; i < recipes.length; i++) {
 const ingredients = recipes[i].ingredients;

 for (let j = 0; j < ingredients.length; j++) {
  if (ingredients[j].ingredient.toLowerCase().includes(query)) {
   result.push(recipes[i]);
   break;
  }
 }
}
```

finished

197 k ops/s ± 1.35%

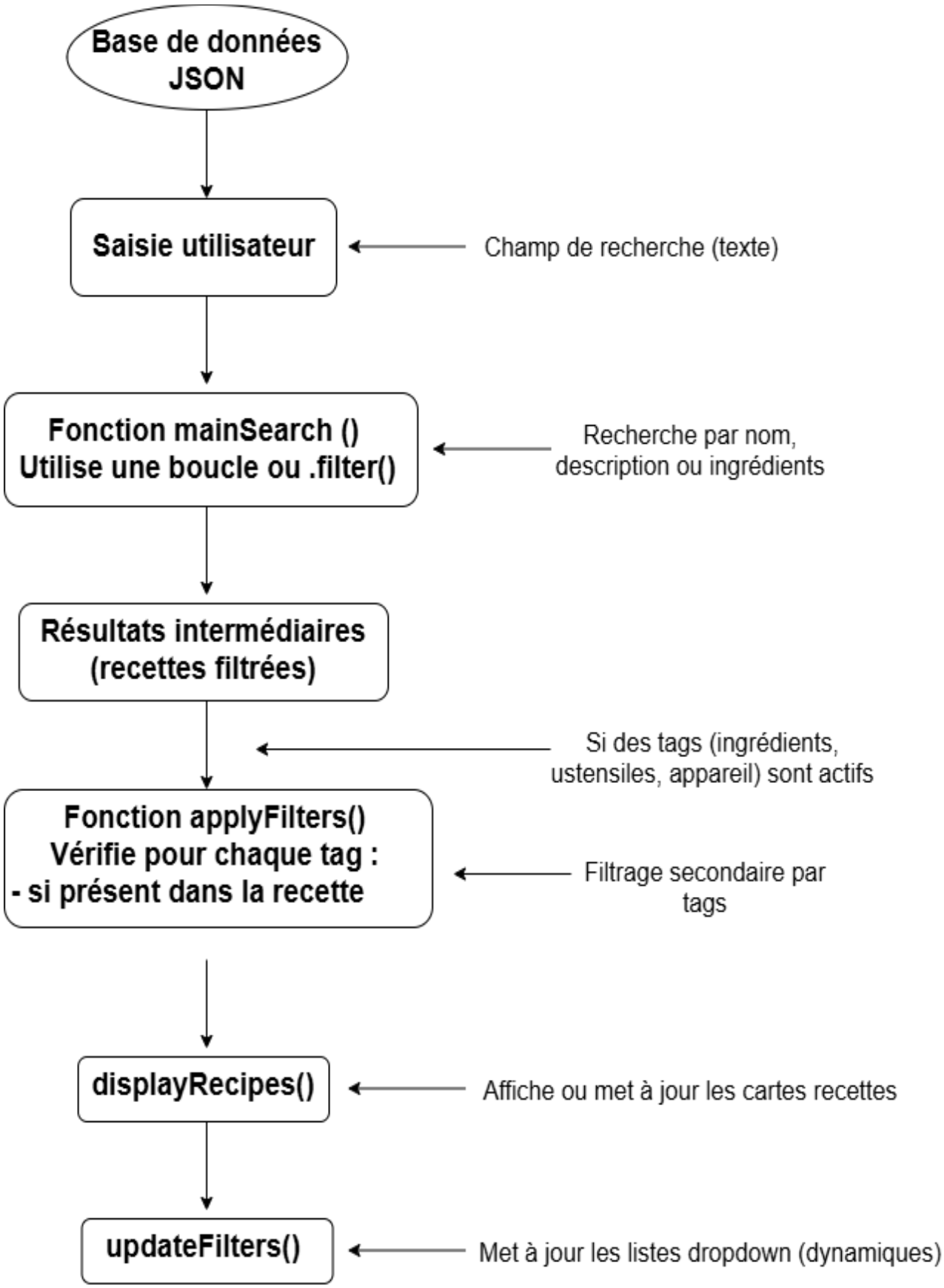
Fastest

Test Case - click to add another test case

Teardown JS - click to add teardown JavaScript

Output (DOM) - click to monitor output (DOM) while test is running

▶ RUN again sort results


Base de données JSON
Saisie utilisateur
Champ de recherche (texte)
Fonction mainSearch ()
Utilise une boucle ou .filter()
Recherche par nom, description ou ingrédients
Résultats intermédiaires (recettes filtrées)
Si des tags (ingrédients, ustensiles, appareil) sont actifs
Fonction applyFilters()
Vérifie pour chaque tag :
- si présent dans la recette
Filtrage secondaire par tags
displayRecipes()
Affiche ou met à jour les cartes recettes
updateFilters()
Met à jour les listes dropdown (dynamiques)