Fiche d’investigation de fonctionnalité

Barre de recherche de recettes

Problématique

Afin de proposer aux utilisateurs un outil de recherche efficace avec des résultats presque instantanés, nous cherchons à obtenir un algorithme de recherche performant.

La barre de recherche doit permettre de rechercher des mots, ou du moins une chaîne de caractères similaire, au sein du titre, de la description des recettes, de l’appareil utilisé, de la liste des ustensiles ou de la liste des ingrédients pour n’afficher que celles correspondantes. Dans les deux implémentations, la vérification se fait de la façon suivante :

- Le nom de la recette contient tout ou partie de la chaîne de caractères saisie ;
- OU la description de la recette contient tout ou partie de la chaîne de caractères saisie ;
- OU l’appareil utilisé dans la recette correspond entièrement ou en partie à la chaîne de caractères saisie ;
- OU la liste des ustensiles nécessaires à la recette contient tout ou partie de la chaîne de caractères saisie ;
- OU la liste des ingrédients contient tout ou partie de la chaîne de caractères saisie.

L’ordre choisi pour la vérification s’explique notamment par les types de données manipulés : le nom, la description et l’appareil sont stockés en tant que simple chaîne de caractères, tandis que les listes d’ustensiles et d’ingrédients sont stockées chacune dans un tableau (contenant lui-même un tableau dans le cas des ingrédients, afin de différencier l’ingrédient lui-même des quantités requises et des unités de mesures correspondantes).

Précisons également que la recherche peut être filtrée par l’utilisateur à l’aide des 3 filtres suivants : *Ingrédients*, *Appareils* et *Ustensiles*. Chacun de ces filtres contient une liste d’éléments correspondants, cette liste étant actualisée à chaque nouvelle recherche de l’utilisateur dans la barre de recherche. En cliquant sur l’un des éléments de la liste au sein de n’importe quel filtre, l’utilisateur peut donc affiner sa recherche en ne retenant qu’un ou un plusieurs ingrédients précis, par exemple. Enfin, il est possible d’utiliser les filtres sans rien saisir dans la barre de recherche.

Pour les besoins du développement, les données sont actuellement stockées dans un fichier JSON faisant office de base de données.

NB : concernant l’analyse comparative, seul l’algorithme permettant le filtrage des recettes par la saisie dans la barre de recherche sera abordé ici.

Analyse comparative

Algorithme de recherche par valeur saisie dans la barre de recherche

Protocole de test

Pour mesurer les performances de chaque algorithme, l'outil d'analyse comparative Perflink a été utilisé en guise de banc d'essai.

Pour les besoins du test, deux tableaux sont initialisés avant l'exécution de chaque algorithme (cette initialisation ne comptant pas dans les résultats) :

- `recipes`, contenant les recettes telles qu'elles ont été écrites dans le fichier JSON ;
- `values`, contenant 20 valeurs susceptibles de figurer dans au moins une des recettes (il peut s'agir d'un titre de recette, d'un ingrédient, d'un appareil ou d'un ustensile).

Les résultats pouvant varier d'un tir au banc à un autre, 10 tirs ont été réalisés sur deux navigateurs différents (Google Chrome et Mozilla Firefox) et une moyenne des résultats (en nombres d'opérations par seconde) a été calculée pour chaque algorithme, afin d'obtenir une comparaison suffisamment pertinente.

Option #1 : Boucles natives

Cette option tire parti des boucles natives proposées par JavaScript (`if`, `for`…). Il s'agit de parcourir le tableau généré depuis le fichier JSON faisant office de base de données à l'aide d'une boucle contenant plusieurs instructions correspondant aux critères énoncés précédemment, afin de générer un second tableau contenant les résultats de la recherche

Avantages	Inconvénients
• Marginalement plus compatible	• Nombre d'instructions assez élevé • Moins performante

Option #2 : Méthodes `Array`

Cette option tire parti des méthodes proposées par l'objet `Array`. Le choix a été fait d'utiliser les méthode `filter` et `find` sur le tableau généré depuis le fichier JSON faisant office de base de données, afin de générer un second tableau contenant les résultats de la recherche filtrés selon les critères énoncés précédemment.

Avantages	Inconvénients
• Une seule instruction pour le filtrage des données • Plus performant	• Marginalement moins compatible

Solution retenue

En moyenne, les performances diffèrent assez largement d’une option à une autre, en faveur de celle se basant sur les méthodes de l’objet `Array`. Cette dernière propose en outre une écriture plus synthétique et élégante, et ne pose pas de souci de compatibilité si le navigateur utilisé est suffisamment récent.

Annexes

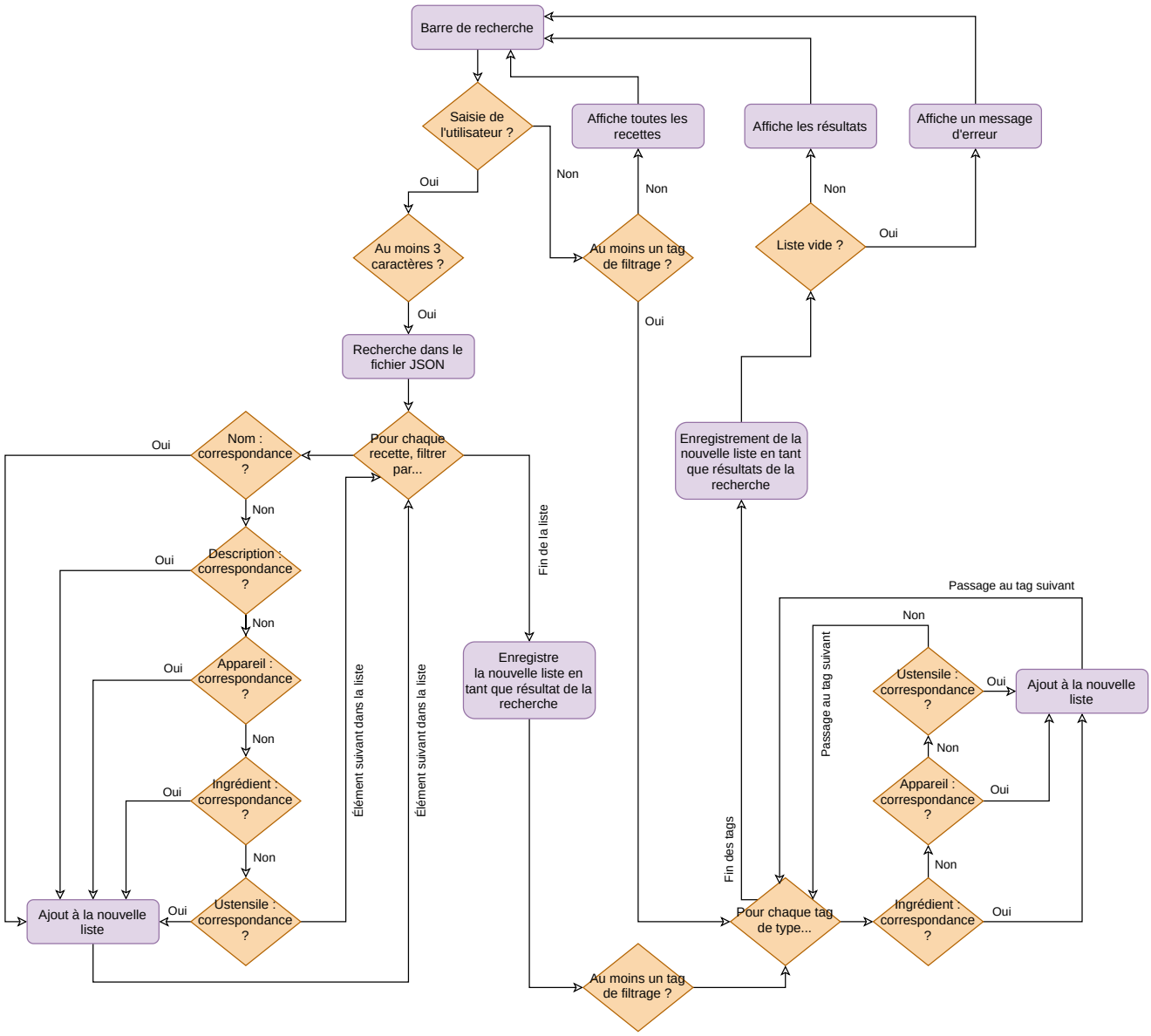

Figure 1 – Flowchart pour la fonctionnalité de recherche

```
const result = []
search_loop: for(let i = 0; i < recipes.length; i++){
  if(recipes[i].name.toLowerCase().includes(value) ||
     recipes[i].description.toLowerCase().includes(value) ||
     recipes[i].appliance.toLowerCase().includes(value)
  ){
    result.push(recipes[i])
    continue
  }
  for(let y = 0; y < recipes[i].utensils.length; y++){
    if(recipes[i].utensils[y].toLowerCase().includes(value)){
      result.push(recipes[i])
      continue search_loop
    }
  }
  for(let x = 0; x < recipes[i].ingredients.length; x++){
    if(recipes[i].ingredients[x].ingredient.toLowerCase().includes(value)){
      result.push(recipes[i])
      continue search_loop
    }
  }
}
return result
```

Figure 2 – Algorithme de recherche par valeur saisie dans la barre de recherche

Extrait du code utilisant les boucles natives

```
const result = recipes.filter(
  (recipe) =>
    recipe.name.toLowerCase().includes(value) ||
    recipe.description.toLowerCase().includes(value) ||
    recipe.appliance.toLowerCase().includes(value) ||
    recipe.utensils.find((utensil) =>
      utensil.toLowerCase().includes(value)
    ) ||
    recipe.ingredients.find((ingredient) =>
      ingredient.ingredient.toLowerCase().includes(value)
    )
)
return result
```

Figure 3 – Algorithme de recherche par valeur saisie dans la barre de recherche

Extrait du code utilisant les méthodes de l'objet `Array`

```
const values = [
  'tartelette',
  'four',
  'beurre',
  'pâte brisée',
  'passoire',
  'blender',
  'couteau',
  'tomate',
  'citron',
  'crème',
  'râpe à fromage',
  'tarte aux pommes',
  'chocolat noir',
  'mozzarella',
  'économe',
  'casserole',
  'vinaigre de cidre',
  'galette bretonne',
  'louche',
  'carbonara'
]
```

Figure 4 – Tableau contenant les termes utilisés pour les tirs au banc

Tir	Boucles natives (ops/s)	Méthodes Array (ops/s)	Delta (%)
1	333	392	17.89
2	346	394	13.87
3	327	385	17.58
4	323	386	19.50
5	331	384	16.01
6	330	393	18.94
7	337	390	15.75
8	350	412	17.57
9	352	411	16.62
10	357	419	17.39
Moyenne	***338***	***396***	***17.09***

Figure 5 – Compilation des résultats des tirs au banc réalisés sur Perflink avec la moyenne pour chaque version de l'algorithme