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).

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

Protocole de test

Pour mesurer les performances de chaque algorithme, l’outil d’analyse JSBench.ch a été utilisé.

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 et une moyenne des résultats 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
• Meilleure compatibilité	• Nombre d'instructions assez élevé • Marginalement moins performant

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 • Légèrement plus performant	• Marginalement moins compatible

Solution retenue

Le choix ne se fera pas sur le plan des performances, très similaires pour les deux algorithmes.

La première option propose la meilleure compatibilité possible au détriment de la simplicité d'écriture ; tandis que la seconde tire parti de méthodes et fonctionnalités plus moderne, permettant certes une implémentation plus synthétique, mais rendant l'utilisation avec des navigateurs non compatibles compliquées sans implémentation des prothèses d'émulation.

Cela étant dit, les méthodes utilisées par la seconde option sont supportées par une grande majorité de navigateurs, pour certains devenus obsolètes. Seules les différentes versions d'Internet Explorer sont dans tous les cas incompatibles, donc à moins d'avoir un intérêt à supporter ces navigateurs, la seconde option est donc retenue.

Annexes

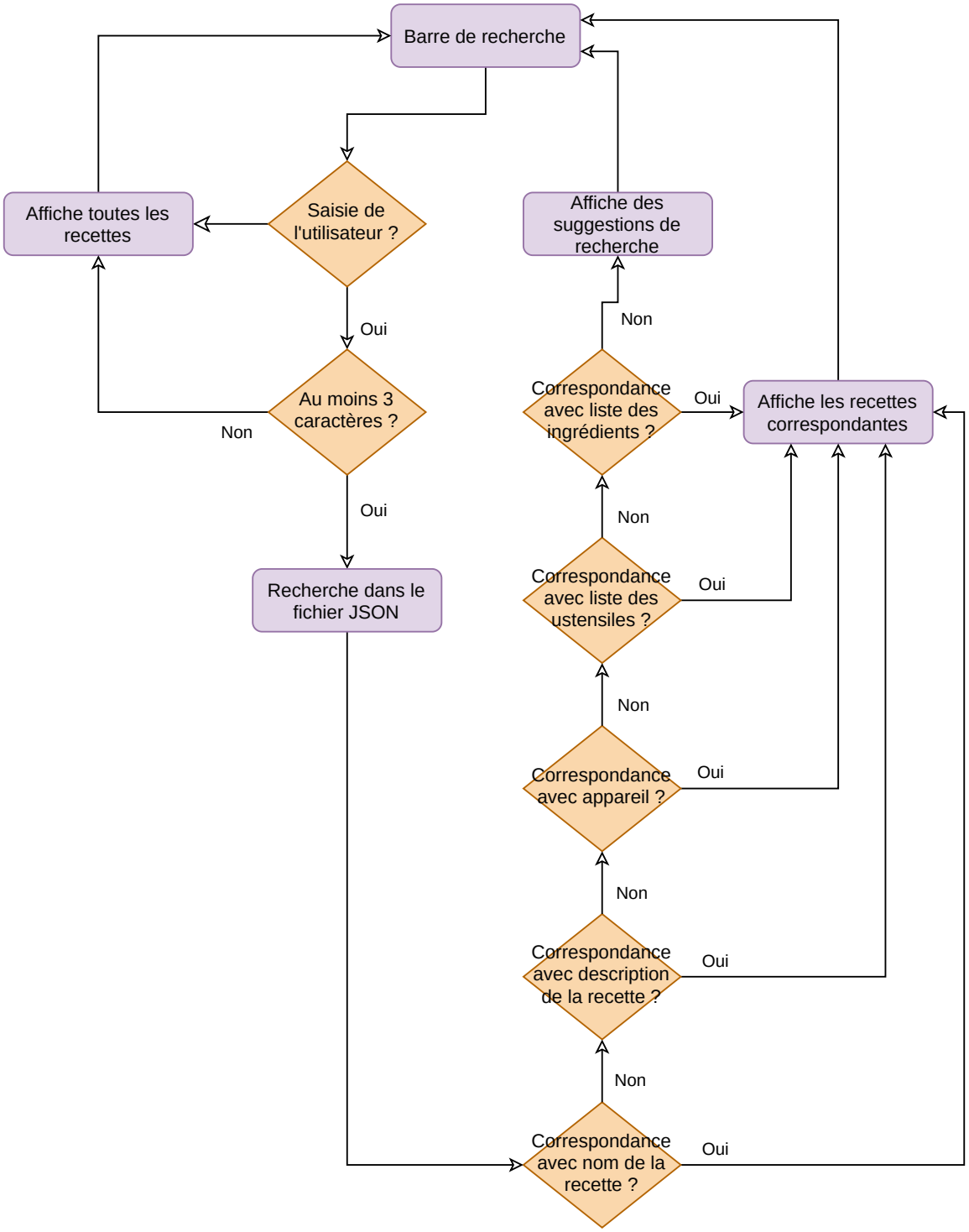

Figure 1 – Flowchart pour la barre de recherche

```
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 2 – Tableau contenant les termes utilisés pour le benchmark

Tirs	Boucles natives	Méthodes Array	Delta
1	1871	1916	2.41 %
2	1878	1933	2.93 %
3	1880	1935	2.93 %
4	1883	1910	1.43 %
5	1892	1938	2.43 %
6	1879	1957	4.15 %
7	1877	1961	4.48 %
8	1874	1935	3.26 %
9	1879	1959	4.26 %
10	1881	1942	3.24 %
Moyennes	***1879.4***	***1938.6***	***3.15 %***

Figure 3 – Tableau compilant le résultat des tirs aux bancs et la moyenne pour chaque algorithme

```
const results = []
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)
  ){
    results.push(recipes[i])
    continue
  }
  for(let y = 0; y < recipes[i].utensils.length; y++){
    if(recipes[i].utensils[y].toLowerCase().includes(value)){
      results.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)){
      results.push(recipes[i])
      continue search_loop
    }
  }
}
return results
```

Figure 4 – 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 5 – Extrait du code utilisant les méthodes `Array`