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 liste des ingrédients ou de la description des recettes, 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).

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 avec des navigateurs plus anciens, voire obsolètes	• 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 • Marginalement plus performant en moyenne	• Écarte les navigateurs non compatibles

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

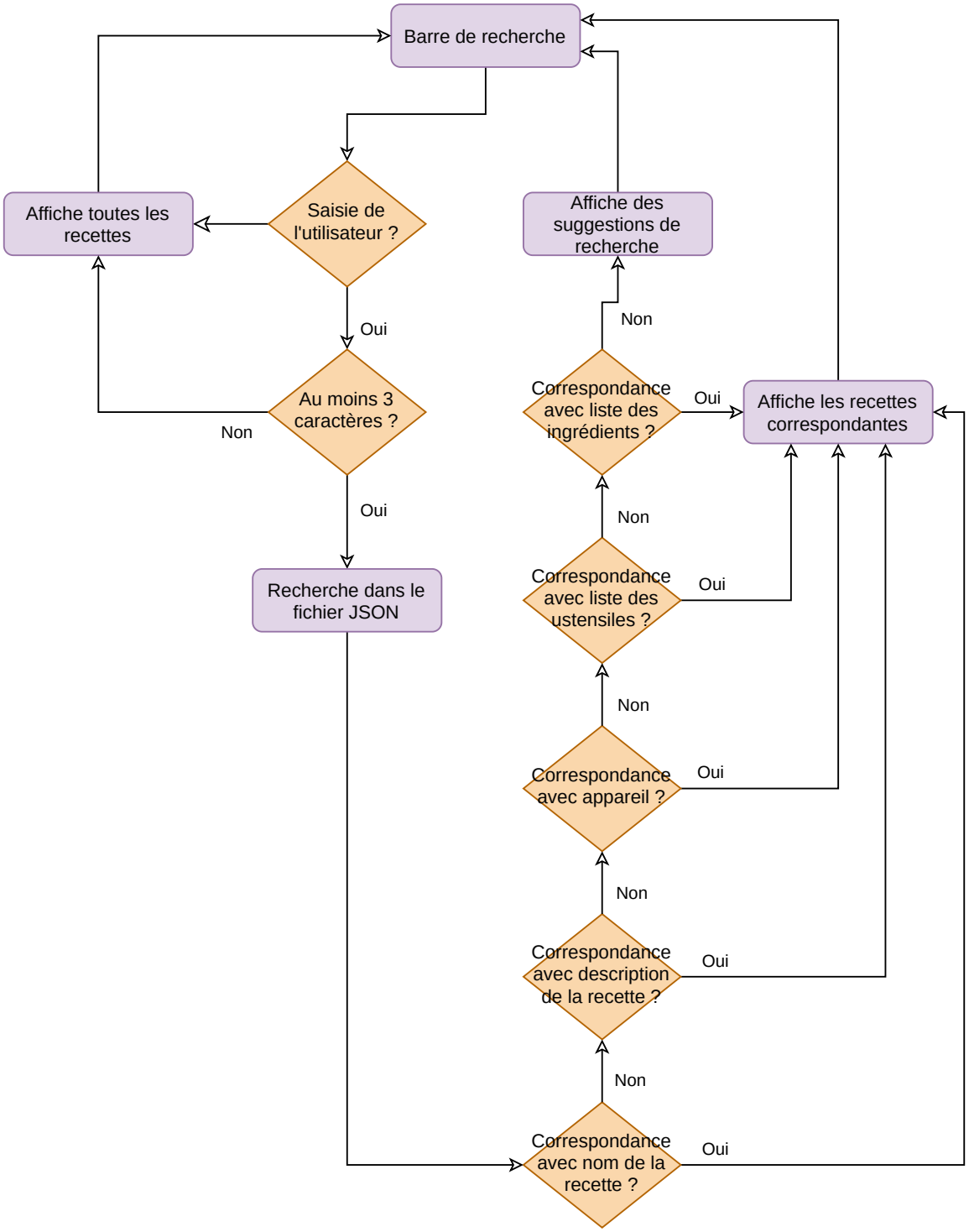

Figure 1 – Flowchart pour la barre de recherche

	Boucles natives	Array
tartelette	6.76	6.32
four	9.16	9.65
beurre	11.11	11.40
pâte brisée	7.81	6.81
passoire	7.30	6.91
blender	8.17	8.62
couteau	13.87	12.00
tomate	9.52	8.87
citron	9.15	8.01
crème	10.52	10.44
râpe à fromage	7.63	8.19
tarte aux pommes	6.33	6.69
chocolat noir	7.14	7.75
mozzarella	5.56	6.84
économe	7.02	7.07
casserole	7.08	6.89
vinaigre de cidre	6.64	7.01
galette bretonne	6.86	4.17
louche	4.77	7.30
carbonara	5.77	6.91
Moyenne (ms)	**7.91**	**7.89**

Figure 2 – Tests de performance (avec `console.time()` *et* `console.timeEnd()`*)*

Note : le résultat pour chaque terme correspond à la durée moyenne sur 5 exécutions consécutives.

```
getSearch() {
  const search = this.search.value.trim().toLowerCase()
  const results = []
  search_loop: for (let i = 0; i < this.recipes.length; i++) {
    if (
      this.recipes[i].name.toLowerCase().indexOf(search) !== -1 ||
      this.recipes[i].description.toLowerCase().indexOf(search) !== -1 ||
      this.recipes[i].appliance.toLowerCase().indexOf(search) !== -1
    ) {
      results.push(this.recipes[i])
      continue
    }
    for (let y = 0; y < this.recipes[i].utensils.length; y++) {
      if (this.recipes[i].utensils[y].toLowerCase().indexOf(search) !== -1) {
        results.push(this.recipes[i])
        continue search_loop
      }
    }
    for (let x = 0; x < this.recipes[i].ingredients.length; x++) {
      if (
        this.recipes[i].ingredients[x].ingredient
          .toLowerCase()
          .indexOf(search) !== -1
      ) {
        results.push(this.recipes[i])
        continue search_loop
      }
    }
  }
  this.displayCards(results)
}
```

Figure 3 – Extrait du code utilisant les boucles natives

```
getSearch() {
  const search = this.search.value.trim().toLowerCase()
  const result = this.recipes.filter(
    (recipe) =>
      recipe.name.toLowerCase().includes(search) ||
      recipe.description.toLowerCase().includes(search) ||
      recipe.appliance.toLowerCase().includes(search) ||
      recipe.utensils.find((utensil) =>
        utensil.toLowerCase().includes(search)
      ) ||
      recipe.ingredients.find((ingredient) =>
        ingredient.ingredient.toLowerCase().includes(search)
      )
  )
  this.displayCards(result)
}
```

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