Fiche d'investigation de fonctionnalité

Fonctionnalité : Recherche	**Fonctionnalité #2**
Problématique : Pour se démarquer de la concurrence nous voulons un algorithme de recherche rapide pour que la recherche paraisse instantanée cf. Figure 4 Flowchart	

Option 1 : approche utilisant les boucles (while, for, forof, forin) cf. Figure.1.1 Code (loop)
Dans cette option la recherche se fait à travers differentes boucles succésive

Avantage	**Incovénients**
• lisible • facilement maintenable/modifiable	• très lent pour la comparaison de string • utilise un buffer de resulat • code plus verbeux (plus long)

- Score JsBench : **3935**/s
- **19** lignes (fonction de comparaison de string inclus)

Option 2 : approche fonctionnelle (foreach, map, filter, find, reduce, ...) cf. Figure 2.1 - Code (functional)
Dans cette option la recherche se fait grâce au fonctions membre des prototype d'objets js

Avantage	**Incovénients**
• comparaison de string très rapide • n'utilise pas de buffer de resulat • code plus court	• `Array.filter()` provoque plus d'instructions que `for..of` • moins facilement maintenable

- Score JsBench : **20699**/s (520% de "loop")
- **10** lignes (52% de "loop")

Option 3 : approche hybrid utilisant les boucles sauf pour la comparaison de string cf. Figure.3.1 Code (hybrid)

Dans cette option la recherche se fait à travers differentes boucles succésive, mais la recherche dans une string se fait via la fonction `String.includes()`

Avantage	**Incovénients**
• baucoups plus rapide • facilement maintenable/modifiable	• utilise un buffer de resulat • code un peu moins lisible

- Score JsBench : **55751**/s (260% de "functional" | 14100% de "loop")
- **12** lignes (+20% de "functional")

Solution retenue :

L'objectif étant la performance, j'ai retenue l'approche "hybrid", qui est de très loin la plus perfomante en plus d'être facilement maintenable.

Annexes

	dataset x 1			dataset x 1000		
query	**"coco"**	**"tarte"**	**"noresult"**	**"coco"**	**"tarte"**	**"noresult"**
Functional	20699	29709	26643	32	31	22
Loop	3935	7313	6073	9	8	6
Hybrid	55751	80104	77884	89	88	69

Figure 0 - Benckmark JsBench

```
function strincludes(str, query) {
  for (let i = 0; i < str.length; i++)
    if (str.substring(i, i + query.length).toLowerCase() === query) return true;
  return false;
}

const results = [];
for (const recipe of recipes) {
  const name = recipe.name.toLowerCase();
  const description = recipe.description.toLowerCase();
  let ingredients = '';
  for (const ingredient of recipe.ingredients)
    ingredients += ` ${ingredient.ingredient.toLowerCase()}`;
  if (
    strincludes(name, query) ||
    strincludes(description, query) ||
    strincludes(ingredients, query)
  )
    results.push(recipe);
}
```

Figure 1.1 - Code (loop)

```
const results = recipes.filter(
  (recipe) ⇒
    recipe.name.toLowerCase().includes(query) ||
    recipe.description.toLowerCase().includes(query) ||
    recipe.ingredients
      .reduce((prev, curr) ⇒ prev.concat(curr.ingredient), [])
      .join(' ')
      .toLowerCase()
      .includes(query)
);
```

Figure 2.1 - Code (functional)

```
const results = [];
for (const recipe of recipes) {
  let ingredients = '';
  for (const ingredient of recipe.ingredients)
    ingredients += ` ${ingredient.ingredient.toLowerCase()}`;
  if (
    recipe.name.toLowerCase().includes(query) ||
    recipe.description.toLowerCase().includes(query) ||
    ingredients.includes(query)
  )
    results.push(recipe);
}
```

Figure 3.1 - Code (hybrid)

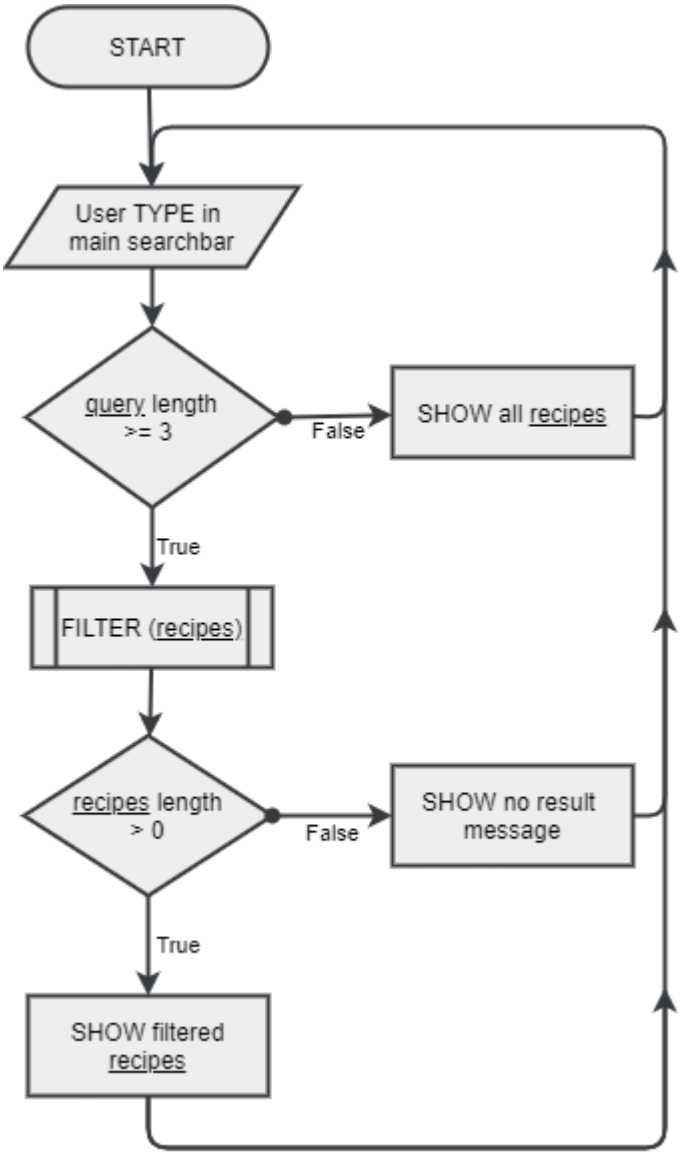

Figure 4 - Flowchart