



# Les petits plats

## Fiche d'investigation de fonctionnalité

### Moteur de recherche principal 🔍



#### Problématique

Trouver l'algorithme de recherche principal le plus performant.

Cet algorithme doit retourner une liste de recette à partir d'une valeur (mots ou groupe de lettres saisis par l'utilisateur dans la recherche principale)

Pour chaque recette retournée la valeur est soit comprise dans son titre, soit ses ingrédients ou bien sa description.

### Option 1: Les méthodes de l'objet Array

Les méthodes de l'objet array sont conçues pour parcourir une liste et transformer chaque membre de cette liste et renvoyer une nouvelle liste ou appliquer une opération à chaque membre de la liste.

#### Les +

- Facile à **Maintenir**
- Facile à **Lire**
- Possibilité d'utiliser le chaînage des méthodes  
🔗 (*method chaining*)

#### Les -

- Transforme l'array d'origine en retournant un nouveau tableau (`array.prototype.filter()`)
- Peut être plus difficile à appréhender pour un débutant

### Option 2: Les boucles (loops)

Elles sont spécialement conçues pour itérer sur une liste tant qu'une condition est vraie

#### Les +

- Plus facile à écrire et à comprendre quand on débute

#### Les -

- Nécessite parfois **beaucoup de mémoire**
- **Plus de ligne** de code

## Solution retenue

Bien que les boucles natives sont plus rapides dans certains cas (les navigateurs et les moteurs JS les ont optimisées pour qu'elles le soient), Le gain de rapidité reste minime (1% Chrome, 8% Firefox selon tests JSBEN.CH)

J’ai donc retenue **les méthodes de l’objet Array** car elle permettent d’avoir un code plus lisible et donc plus facilement maintenable.

Certaines optimisations doivent faire partie des bonnes pratiques générales. Si les performances sont importantes et que les tableaux utilisés sont très volumineux.

Pour aller plus loin, il est préférable d’écrire un test pour vérifier cela puis arbitrer entre l’usage de l’option 1 ou l’option 2 en sachant qu’il s’agit d’une **solution temporaire** puisqu’à terme l’algorithme sera côté backend.

## Ressources

### L’objet global array

Array - JavaScript | MDN

L'objet global Array est utilisé pour créer des tableaux. Les tableaux sont des objets de haut-niveau (en


 [https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global\\_Objects/Array](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Array)




### La boucle for..of

for...of - JavaScript | MDN


L'instruction for...of permet de créer une boucle Array qui parcourt un objet itérable (ce qui inclut les objets

 <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/for...of>





### Method chaining

Utiliser une ligne de code en enchaînant les méthodes à la suite

JavaScript method chaining 

JavaScript method chaining tutorial example explained#javascript #method #chaining// \*\*\*\*\*

 <https://www.youtube.com/watch?v=JA84lnOHcDM>




### Loops vs Array Methods

Article intéressant qui m’a permis d’être plus pertinent dans mon choix

Loops vs Array Methods

After researching when and how to use array methods like .map, .forEach, and .reduce I found that the performance of a for-loop and .map() function that produce the same output and act on the same input,

 <https://medium.com/@gabriellegianna92/loops-vs-array-methods-26999051ba45#:~:text=For%20loops%20are%20a%20looping,operation%20to%20each%20list%20member>

### Exemple dans le code

```
// with loop method (without chaining) => 8 lines
const isFound = (array, property, value) => {
  for (const item of array) {
    if (isIncluded(item[property], value)) {
      return true
    }
  }
  return false
}

// with Array methods (Chaining) => 1 line
const isFound = (array, property, value) => array.find(item => isIncluded(item[property], value))
```

# Annexes

## Algorigramme

<https://www.figma.com/file/srk7c9ZvgheWzPwET0G7IT/Algorigramme-Les-petits-plats?node-id=1%3A585>

## Comparatif jsben.ch

Sur Chrome: **Option 1 est la plus rapide** ⇒ néanmoins, j’ai testé plusieurs fois et sur plusieurs navigateurs et cela peut varier sans jamais afficher des résultats trop disparate.

JSBEN.CH

BENCHMARKBROWSEDONATE

Setup block (useful for function initialization. It will be run before every test, and is not part of the benchmark.)

```
1  const data = {
2    {
3      "id": 1,
4      "name" : "Limonade de Coco",
5      "servings" : 1,
6      "ingredients": [
7        {
8          "ingredient" : "Lait de coco",
9          "quantity" : 400,
10         "unit" : "ml"
11       },
12       {
13         "ingredient" : "Jus de citron",
```

ADD LIBRARY

boilerplate block (code will executed before every block and is part of the benchmark. use it for data initializing.)

code block 1

1 filterMainSearch(data, value);

code block 2

1 filterMainSearchBis(data, value);

code block 1 (80505) 🏆

100%

code block 2 (79643)

98.93%

DON'T UPDATE TO WINDOWS 11!

CLICK HERE

If you like to donate (Thank you!):

Ethereum (ETH)

Chia (XCH)

Cardano (ADA)

Ravencoin (RVN)

Bitcoin (BTC)

Ripple (XRP)

Litecoin (LTC)

Monero (XMR)

Dogecoin (DOGE)

SOLANA (SOL)