

# Fiche d'investigation de fonctionnalité

Fonctionnalité: Search recipes	Fonctionnalité #2
<b>Problématique</b> : Develop best performing algorithm t	to research recipes.

Option 1: Develop search algorithm using built-in methods of array objects (i.e. filter)

This option will concentrate primarily on utilisation of array method available in ES6 'out-of-the-box'.

# **Avantages:**

- Concise and easy to read and to maintain
- Faster code parsing if no actual filtering is performed (cf. Annex B)
- Performance increases with scale since filter()
  creates its own "asynchronous" call stack in the
  event loop (cf. Annex D vs Annex C)

#### Inconvénients:

• Less performant with smaller datasets due to function overhead

#### **Observations & comments:**

In development, the Array.filter() method produces cleaner code. In execution, it is somewhat faster than the for statement at 533 193 ops/s vs 503 317 ops/s with no actual data to filter (cf. Annex B), yet it is 99% as fast with the sample data set of 50 recipes (cf. Annex C). That said, the Array.filter() method scales better with data size increases due to "asynchronous" nature of its call stack in the event loop which results in 544 ops/s vs 526 ops/s for for statement with 2 000 recipes.

**Option 2**: Develop search algorithm using native loops (e.g. while, for)

This option will concentrate primarily on hand coding search algorithm using procedural statements.

# **Avantages:**

 Faster performance with smaller datasets (cf. Annex C)

### Inconvénients:

- The code is harder to read and to maintain
- Slower performance at code parsing when no actual filtering is performed (cf. Annex B)
- Performance decreases with scale because looping through data will block any other task that needs to run on the main thread (cf. Annex D)

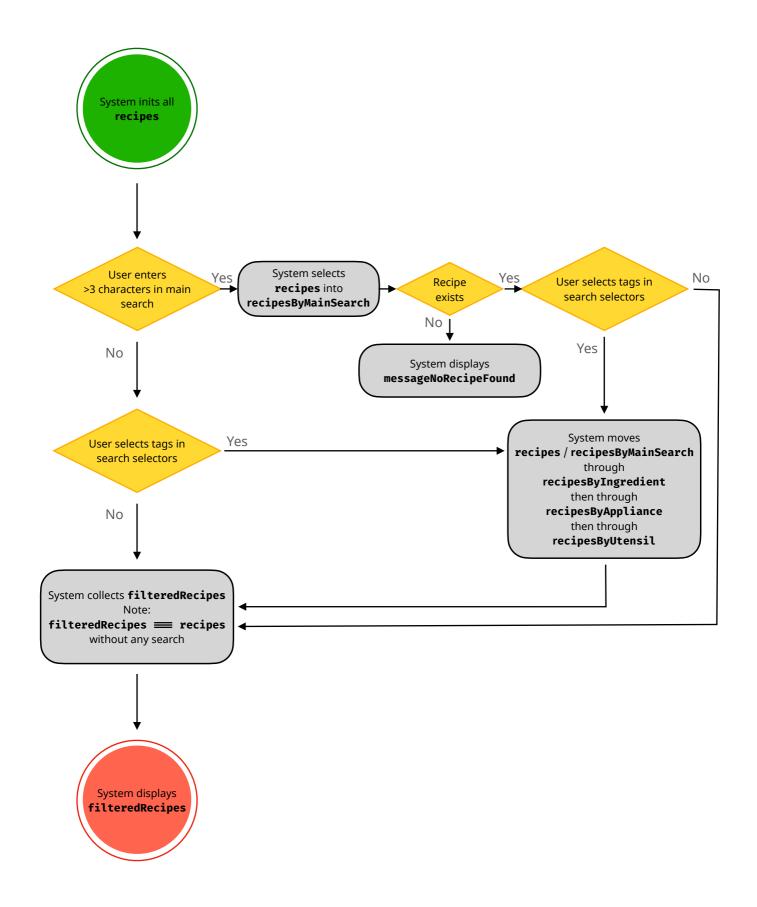
#### **Observations & comments:**

In development, the for statement is verbose and less clear. In execution of the sample data set of 50 recipes it performed faster than Array.filter() at 25 284 ops/s vs 24 995 ops/s. That said, this performance decreases with scale because for-loops are synchronously run and will therefore block all other tasks until finished. In execution of the sample data set of 2 000 recipes it slowed down relative to Array.filter() at 526 ops/s vs 544 ops/s.

## **Retained solution:**

Considering the needs for [1] readability / maintainability, [2] performance and [3] scalability of the retained solution, I recommend to prioritize the Array.find() method. While it is marginally slower than the for statement with the available 50 recipe data sample, its performance scales up much better as recipe data grows. With 2 000 recipes, the for statement is just 97% as efficient as the Array.filter() method. In addition, the Array.find() method is much easier to read and therefore maintain.

# Annex A: Search Flow Diagram





JSBEN.CH		
Search performance for 50 recipes (inputMainSearch = "") □		
Setup block (useful for function initialization. it will be run before every test, and is not part of the benchmark.)		
boilerplate block (code will executed before every block and is part of the benchmark. use it for data initializing.)	~	
Array.filter()		
<pre>let recipesByMainSearch = recipes.filter(recipe =&gt;</pre>		
for statement		
<pre>let recipesByMainSearch = []; for (let i = 0; i &lt; recipes.length; i++) {</pre>		
RUN TESTS GENERATE PAGE URL	NEW BENCHMARK	
result		
Array.filter() (533193) 🏆		
100%		
for statement (503317)		
94.4%		



Search performance for 50 recipes (inputMainSearch = "ROUGE")   Setup block (costal for function initialization, it will be included every test, and is not part of the bordwhark)  **  **  **  **  **  **  **  **  **	JSBEN.CH		
Dollerplate block (code will execused before every block and is part of the benchmark use 8 for data initiations)  Array filter() □  1	Search performance for 50 recipes (inputMainSearch = "ROUGE")		
Arrayfilter()     1	Setup block (useful for function initialization. it will be run before every test, and is not part of the benchmark.)	~	
let recipesByMainSearch = recipes.filter(recipe =>   recipe.name.toUpperCase().includes(inputMainSearch)      recipe.description.toUpperCase().includes(inputMainSearch)      recipe.ingredients.some(detail => detail.ingredient.toUpperCase().includes(inputMainSearch)      recipesByMainSearch = [];   for (let i = 0; i < recipes.length; i++) {   if (recipes[1].name.toUpperCase().includes(inputMainSearch)      a recipes[i].description.toUpperCase().includes(inputMainSearch)      5	boilerplate block (code will executed before every block and is part of the benchmark. use it for data initializing.)	<b>~</b>	
<pre>recipe.name.toUpperCase().includes(inputMainSearch)    recipe.description.toUpperCase().includes(inputMainSearch)    recipe.ingredients.some(detail =&gt; detail.ingredient.toUpperCase().includes(input    let recipesByMainSearch = [];   for (let i = 0; i &lt; recipes.length; i++) {   if (recipes[i].name.toUpperCase().includes(inputMainSearch)      recipes[i].description.toUpperCase().includes(inputMainSearch)     </pre>	Array.filter()		
<pre>let recipesByMainSearch = { ]; for (let i = 0; i &lt; recipes.length; i++) {     if (recipes[i].name.toUpperCase().includes(inputMainSearch)   </pre>	recipe.name.toUpperCase().includes(inputMainSearch)    3 recipe.description.toUpperCase().includes(inputMainSearch)		
<pre>for (let i = 0; i &lt; recipes.length; i++) {     if (recipes[i].name.toUpperCase().includes(inputMainSearch)   </pre>	for statement		
result  for statement (25284)   100%  Array.filter() (24995)	<pre>for (let i = 0; i &lt; recipes.length; i++) {     if (recipes[i].name.toUpperCase().includes(inputMainSearch)            recipes[i].description.toUpperCase().includes(inputMainSearch)            recipes[i].ingredients.some(detail =&gt; detail.ingredient.toUpperCase().includes</pre>		
for statement (25284) <b>*</b> 100%  Array.filter() (24995)	RUN TESTS GENERATE PAGE URL	NEW BENCHMARK	
100% Array.filter() (24995)	result		
Array.filter() (24995)	for statement (25284) 🏆		
2 2 2 2 2	100%		
98.86%	Array.filter() (24995)		



JSBEN.CH
Search performance for 2 000 recipes (inputMainSearch = "ROUGE") □
Setup block (useful for function initialization. it will be run before every test, and is not part of the benchmark.)
boilerplate block (code will executed before every block and is part of the benchmark. use it for data initializing.)
Array.filter()
<pre>1  let recipesByMainSearch = recipes.filter(recipe =&gt; 2</pre>
for statement
<pre>1 let recipesByMainSearch = []; 2</pre>
RUN TESTS GENERATE PAGE URL NEW BENCHMARK
result
Array.filter() (544) ♥
100%
for statement (526) 96.69%
70.07%