Les petits plats

Fiche d'investigation de fonctionnalité

Fonctionnalité: Search recipes	**Fonctionnalité #2**
Problématique: Develop best performing algorithm 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:	**Inconvénients**:
• 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)	• 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:	**Inconvénients**:
• Faster performance with smaller datasets (cf. Annex C)	• 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:

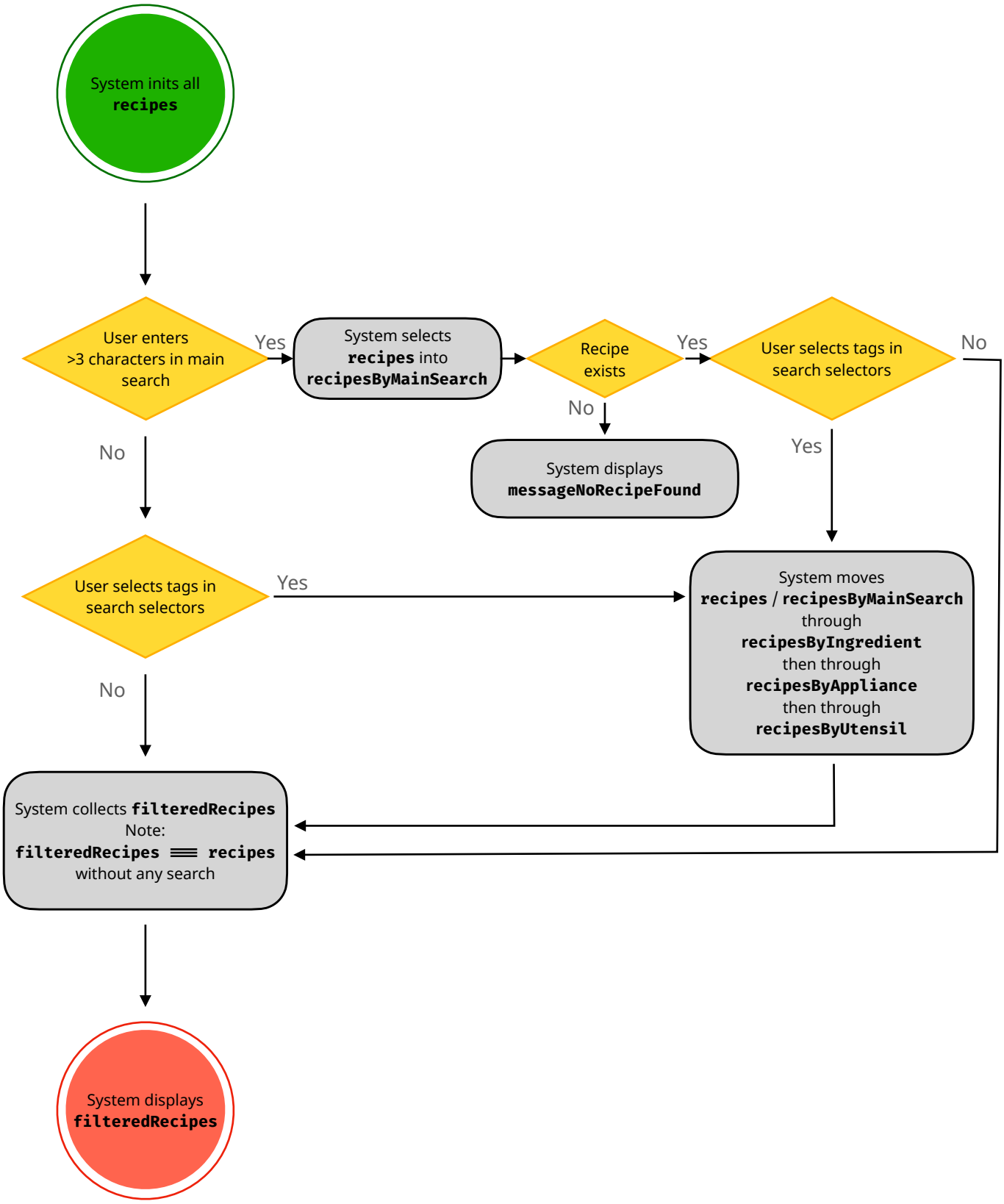
In development, the `for statement` is verbose and less clear. In execution of the sample data set of 50 recipes it performed faster than `Array.filter()` at 25 284 ops/s vs 24 995 ops/s. That said, this performance decreases with scale because for-loops are synchronously run and will therefore block 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:

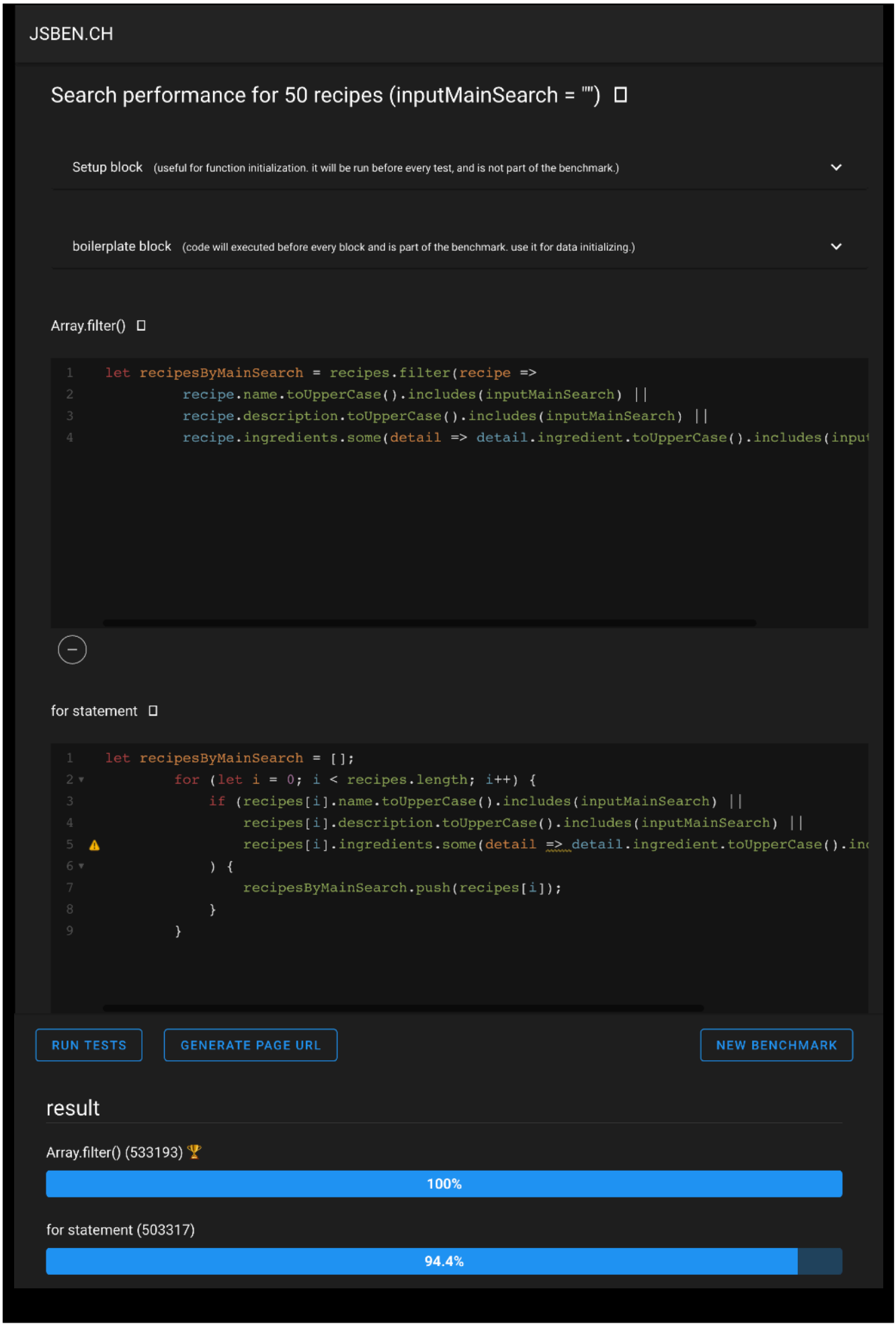
Considering the needs for [1] readability / maintainability, [2] performance and [3] scalability of the retained solution, I recommend to prioritize the `Array.find()` method. While it is marginally slower than the `for statement` with the available 50 recipe data sample, its performance scales up much 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.

Les petits plats

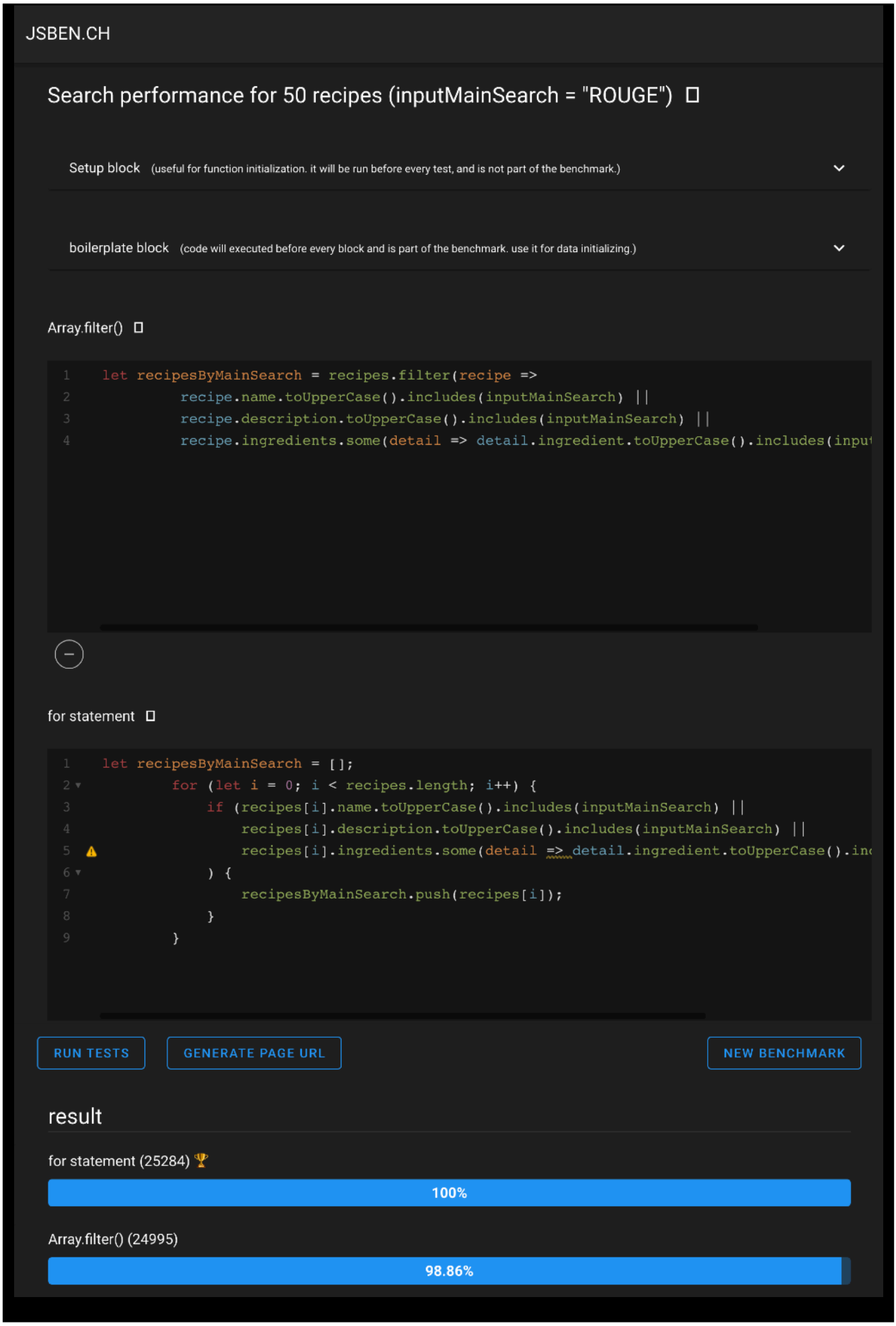
Annex A: Search Flow Diagram



Annex B: Function Test Report (1)



Annex C: Function Test Report (2)



Annex D: Function Test Report (3)