Functionality Investigation Sheet

Feature: Filter Recipes in the User Interface	**Feature #3**

Problem: Quickly access a recipe corresponding to a user's request among recipes in database. The user should be able to filter recipes along two axes: a main bar to search for words or groups of letters in the title, ingredients, or description and a search by keywords in ingredients, utensils, or appliance. The following analyze is concerning the main bar search, providing 2 algorithms and a final proposal for the most efficient one.

Option 1: First search algorithm
In this option the search algorithm takes the user input from the main search bar and filter recipes accordingly. The search function searches each name and description fields, and also iterates the ingredients one by one and repeats the operation with all the recipes found in the data base. If any of the above field matches with the text in the input field, then the recipe is added to the list of the recipes that have to be displayed.

Benefits	**Disadvantages**
⊕ The algorithm works correctly ⊕ Easy to use in case other searching fields are added to the recipe ⊕ Doesn't require preparation for other fields ⊕ Lower memory footprint	⊖ more complicated algorithm because it has to match on multiple fields ⊖ Lower performance comparing to the other search algorithm developed

Minimum number of characters in the main input: 3
Search within name, ingredients and description field

Option 2: Second search algorithm
In this option, after the user load the recipe page, the search function is creating a string with searchable fields (name, ingredients, description) for each recipe. After user types in the main search field, the search function will check the input in the search string of each recipe and display the matching recipes.

Benefits	**Disadvantages**
⊕ Better performance comparing to the first search algorithm ⊕ When input "lemon", second algorithm had a better performance by 79% ⊕ When input "abc" (this input shouldn't match any recipe), second algorithm had a better performance by 73% ⊕ When input "ab" (minimum search input is 3 char). Second algorithm had a better performance by 73%	⊖ ⊖ requires additional preparation and memory when the page is loaded

Minimum number of characters in the main input: 3
Creates string with searchable for each recipe
Displays recipes matching the search input

Solution Retention:
I propose to use the second search algorithm. The reason is that it performs better in all the test: regular input, non-matching input, ad 2 char input. The code is simpler, more explicit and easily to connect to the backend.

Annexes

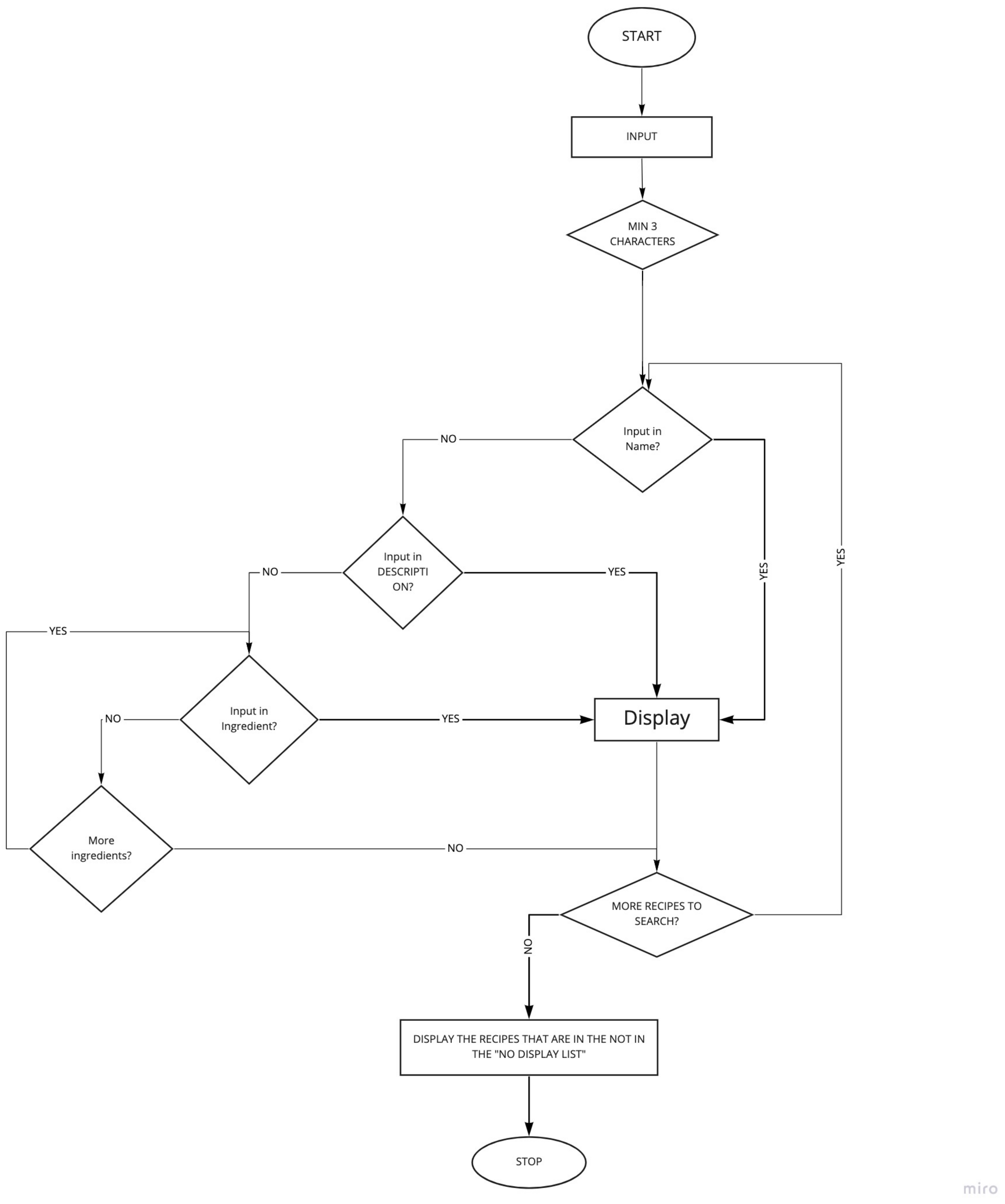


Figure 1 – Search algorithm 1

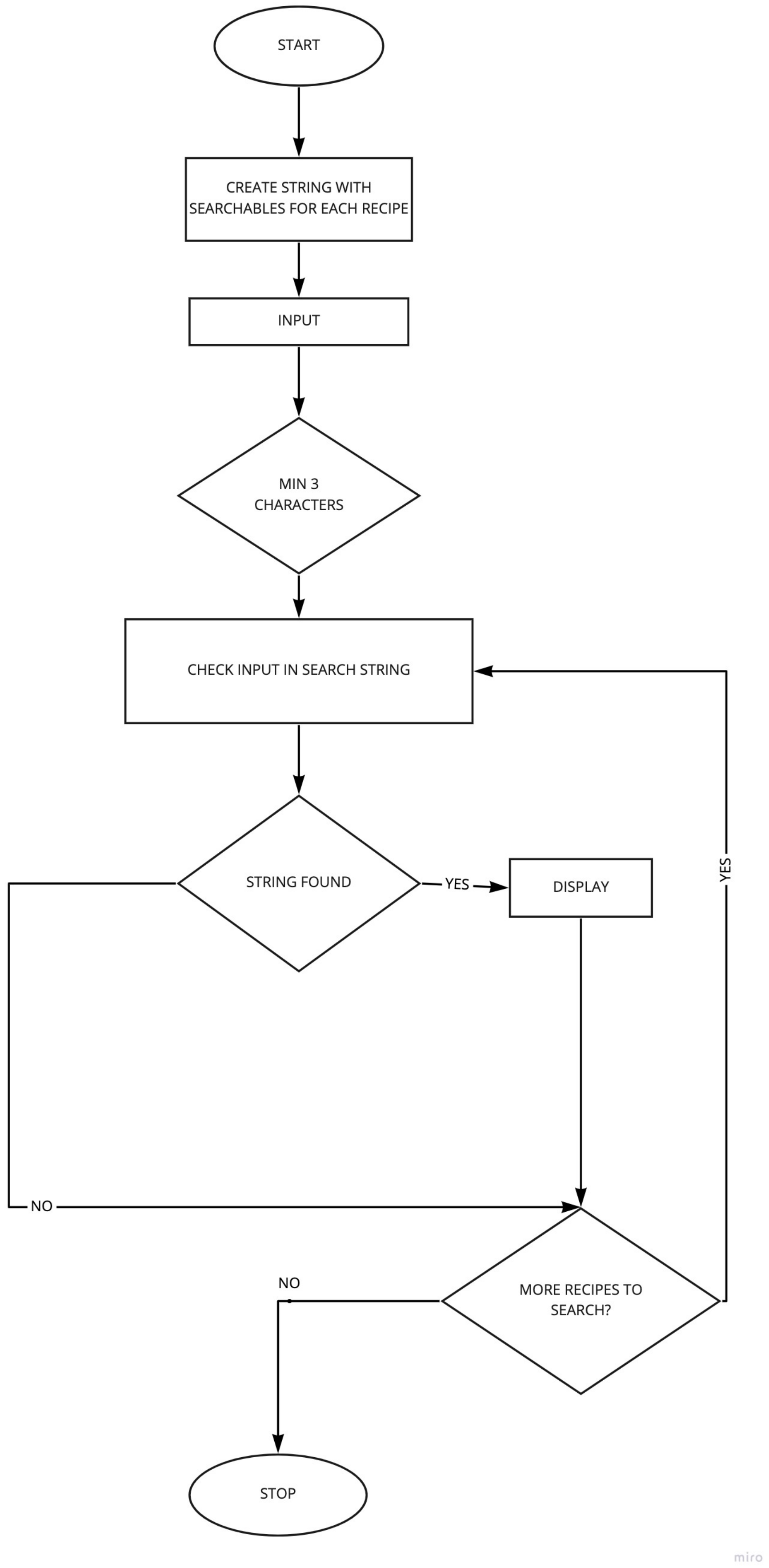
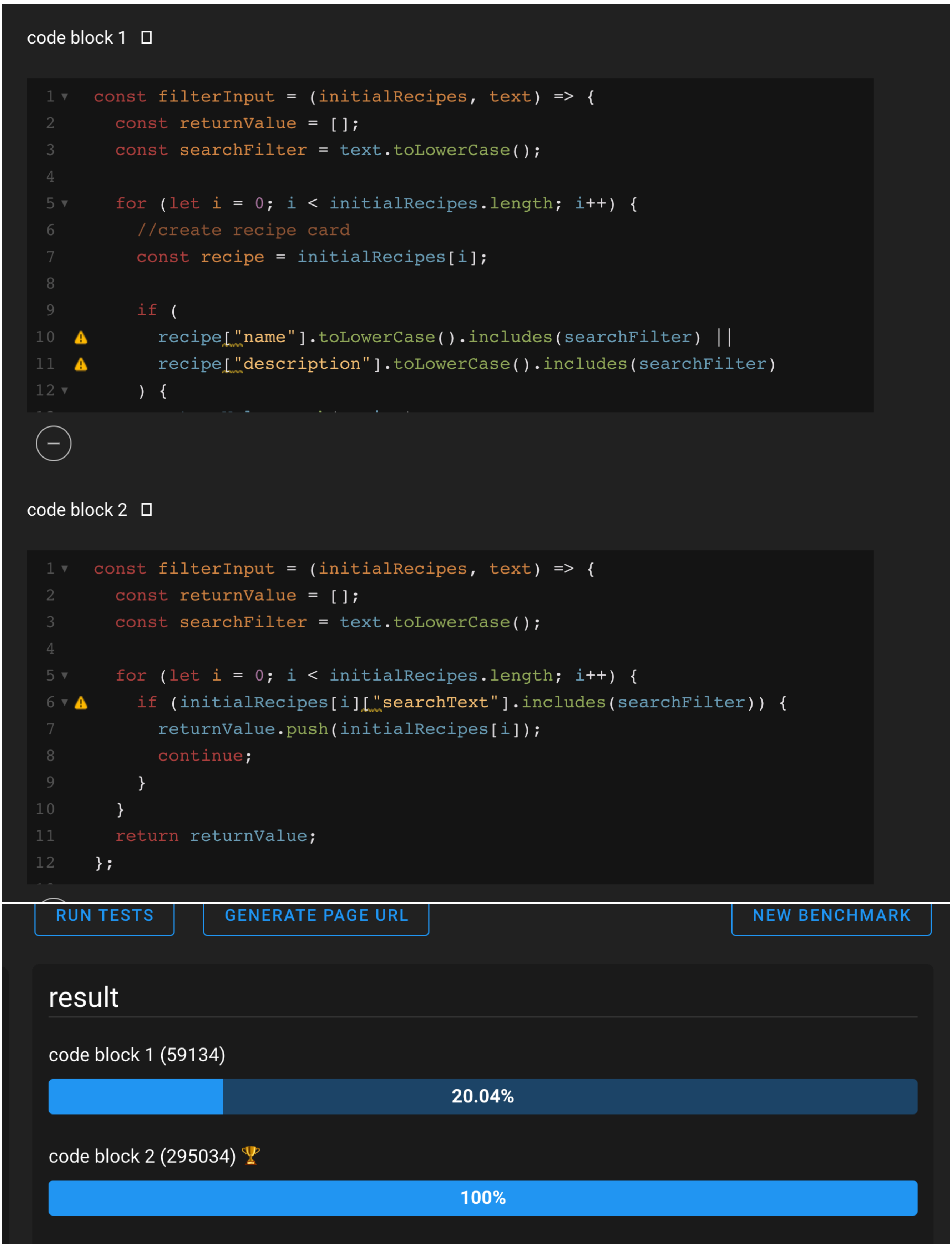


Figure 2 - Search algorithm 2

Algorithm Comparison results with jsben.ch
Comparison 1: search input “lemon”:



code block 1

```
const filterInput = (initialRecipes, text) => {
  const returnValue = [];
  const searchFilter = text.toLowerCase();

  for (let i = 0; i < initialRecipes.length; i++) {
    //create recipe card
    const recipe = initialRecipes[i];

    if (
      recipe["name"].toLowerCase().includes(searchFilter) ||
      recipe["description"].toLowerCase().includes(searchFilter)
    ) {
```

code block 2

```
const filterInput = (initialRecipes, text) => {
  const returnValue = [];
  const searchFilter = text.toLowerCase();

  for (let i = 0; i < initialRecipes.length; i++) {
    if (initialRecipes[i]["searchText"].includes(searchFilter)) {
      returnValue.push(initialRecipes[i]);
      continue;
    }
  }
  return returnValue;
};
```

result

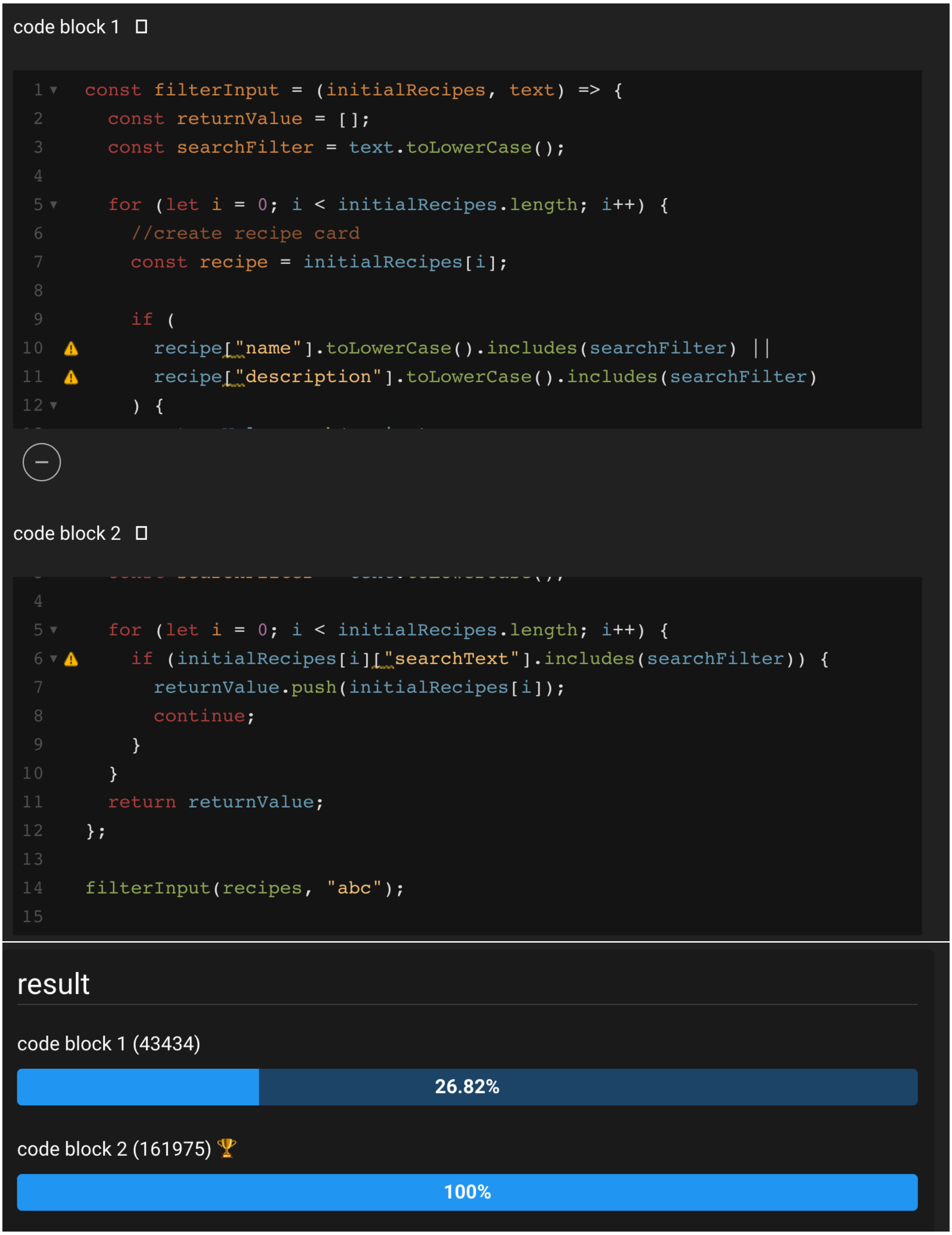
code block 1 (59134)

20.04%

code block 2 (295034) 🏆

100%

Comparison 2: search input “abc”, (no recipe contains this string):



code block 1

```
const filterInput = (initialRecipes, text) => {
  const returnValue = [];
  const searchFilter = text.toLowerCase();

  for (let i = 0; i < initialRecipes.length; i++) {
    //create recipe card
    const recipe = initialRecipes[i];

    if (
      recipe["name"].toLowerCase().includes(searchFilter) ||
      recipe["description"].toLowerCase().includes(searchFilter)
    ) {
```

code block 2

```

  for (let i = 0; i < initialRecipes.length; i++) {
    if (initialRecipes[i]["searchText"].includes(searchFilter)) {
      returnValue.push(initialRecipes[i]);
      continue;
    }
  }
  return returnValue;
};

filterInput(recipes, "abc");

```

result

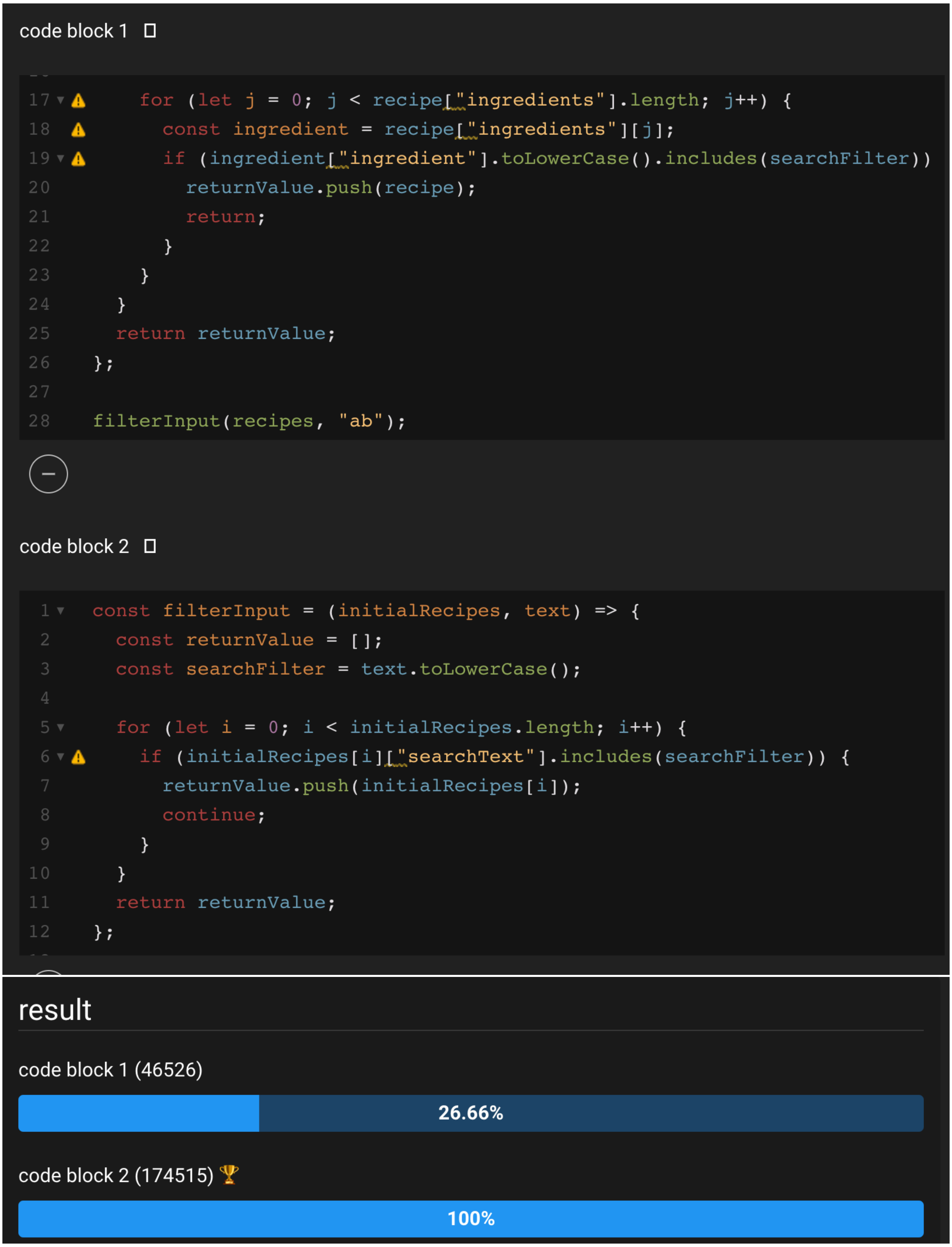
code block 1 (43434)

26.82%

code block 2 (161975) 🏆

100%

Comparison 3: search input “ab”, (input must have at least 3 characters):



code block 1

```javascript
    for (let j = 0; j < recipe["ingredients"].length; j++) {
      const ingredient = recipe["ingredients"][j];
      if (ingredient["ingredient"].toLowerCase().includes(searchFilter))
        returnValue.push(recipe);
        return;
      }
    }
  }
  return returnValue;
};

filterInput(recipes, "ab");
```

code block 2

```javascript
const filterInput = (initialRecipes, text) => {
  const returnValue = [];
  const searchFilter = text.toLowerCase();

  for (let i = 0; i < initialRecipes.length; i++) {
    if (initialRecipes[i]["searchText"].includes(searchFilter)) {
      returnValue.push(initialRecipes[i]);
      continue;
    }
  }
  return returnValue;
};
```

result

code block 1 (46526)

26.66%

code block 2 (174515) 🏆

100%