

Fiche d'investigation de fonctionnalité

Fonctionnalité : La recherche de recettes	Fonctionnalité #1
Problématique : Dans le but de proposer aux clients, une recherche rapide et qui convient parfaitement aux attentes de l'utilisateur, nous cherchons à avoir un programme élaboré la plus rapide,convenable et fonctionnel. Cela nous permet de trier parmi les recettes accessibles via un champ de recherche ainsi que des menus de sélections : d'ingrédients, d'appareils et d'ustensiles. La recherche renverra une nouvelle liste des recettes correspondante avec le nom, ou la description ou les ingrédients.	

Option 1 : Programmation Moderne algo-1

Il filtre le tableau des recettes et renvoie les recettes qui ont l'entrée dans leur nom, ingrédients ou descriptif.L'utilisation des méthodes de l'objet Array (filter, includes, la méthode some() etc...)
Quand l'utilisateur rentre une valeur de l'input, cette dernière, si supérieur de 3 caractères, on l'envoie vers une condition, qui se trouve à l'intérieur d'un filter du tableau avec toute la liste des recettes. Dans cette condition, on créera un autre tableau qui contient juste, les recettes dont on a une correspondances trouvées dans le nom, la description ou les ingrédients et qui servira ensuite pour l'affichage de nos recettes.
La liste des ingrédients, ustensiles et appareils, sera mise à jour grâce à ce tableau trié.

Les avantages Le code est plus facile à comprendre.
Et il reste stable et rapide.

Les inconvénients
Cela réduire la lisibilité du code.

Saisie de 3 caractères minimum dans le champ de recherche principal

Option 2 : Programmation native algo-2

Utilisation des boucles (for ...)
Ici, il prend un tableau de recettes et une chaîne d'entrée, et renvoie un tableau de recettes qui correspondent à la chaîne d'entrée.
Si oui, la recette en question est ajoutée à un nouveau tableau qui servira à l'affichage des recettes trouvées.
La liste des ingrédients, des ustensiles et des appareils, sera mise à jour grâce à ce tableau trié.

Les avantages
Le code est plus optimisé

Les inconvénients
Le code est un peu plus compliqué à lire

Saisie de 3 caractères minimum dans le champ de recherche principal

La solution retenue

Type your text

Tests effectués sur 50 recettes:

Test avec JsBench.me :

La programmation moderne semble être la plus rapide d'après le test Jsbench (Voir Annexe 1 : Test jsBench.me)

La programmation native reste légèrement moins rapide de 0,77% par rapport à la programmation moderne.
avec un slower de 59073%.

- Tests avec jsben.ch :

La programmation native toujours moins rapide (Annexe 2 : Test jsBench.ch)

La programmation moderne toujours plus rapide par rapport à la programmation native (Annexe 2 : Test jsBench.ch)

Le choix retenu

L'option 1, la programmation moderne est retenue car elle est beaucoup plus performante.

(Annexe 1 : Test jsBench.me)

The screenshot shows the jsBench.me website interface. The top navigation bar includes the JSBench.Me logo, a 'Run' button, a prompt to 'Please login/register to save & publish tests', and links for 'Sponsor', 'Settings', and 'Sign in'. The main content area is divided into two panels. The left panel displays the test status as 'finished' with a performance result of '188252.09 ops/s ± 0.77%' and a note '59.73 % slower'. The right panel shows the JavaScript code for the benchmark, which includes a search algorithm and a list of recipes. The code is as follows:

```
soir pour le lendemain. Après avoir laissé mariner le poisson, coupez le concombre en fines rondelles sans la peau et les tomates en prenant soin de retirer les pépins. Rayer la carotte. Ajouter les légumes au poissons avec le citron cette fois ci dans un Saladier. Ajouter le lait de coco. Pour ajouter un peu plus de saveur vous pouvez ajouter 1 à 2 cuillères à soupe de Crème de coco",
"appliance": "Saladier",
"ustensils": ["Presse citron"]
}
};

const searchAlgo = (recipes, input) => {
  const list = [];
  for (let i = 0; i < recipes.length; i++) {
    const recipe = recipes[i];
    const recipeName = recipe.name.toLowerCase();
    const recipeDescription = recipe.description.toLowerCase();
    const regex = new RegExp("(?:" + input.toLowerCase() + ")", "i");
    const isNameMatch = regex.test(recipeName);
    const isDescriptionMatch = regex.test(recipeDescription);

    for (let i = 0; i < recipe.ingredients.length; i++) {
      const ingredients = recipe.ingredients[i];

      const isIngredientMatch = regex.test(ingredients.ingredient);

      if (isIngredientMatch) {
        list.push(recipe);
      }
    }
    if (isNameMatch) {
      list.push(recipe);
    }
    if (isDescriptionMatch) {
      list.push(recipe);
    }
  }
}

const searchAlgo = (recipes, input) => {
  return recipes.filter(recipe => {
    const list = [];
    for (let i = 0; i < recipes.length; i++) {
      const recipe = recipes[i];
      const recipeName = recipe.name.toLowerCase();
      const recipeDescription = recipe.description.toLowerCase();
      const regex = new RegExp("(?:" + input.toLowerCase() + ")", "i");
      const isNameMatch = regex.test(recipeName);
      const isDescriptionMatch = regex.test(recipeDescription);

      for (let i = 0; i < recipe.ingredients.length; i++) {
        const ingredients = recipe.ingredients[i];

        const isIngredientMatch = regex.test(ingredients.ingredient);

        if (isIngredientMatch) {
          list.push(recipe);
        }
      }
      if (isNameMatch) {
        list.push(recipe);
      }
      if (isDescriptionMatch) {
        list.push(recipe);
      }
    }
  });
};

const searchAlgo = (recipes, input) => {
  return recipes.filter(recipe => {
    const list = [];
    for (let i = 0; i < recipes.length; i++) {
      const recipe = recipes[i];
      const recipeName = recipe.name.toLowerCase();
      const recipeDescription = recipe.description.toLowerCase();
      const regex = new RegExp("(?:" + input.toLowerCase() + ")", "i");
      const isNameMatch = regex.test(recipeName);
      const isDescriptionMatch = regex.test(recipeDescription);

      for (let i = 0; i < recipe.ingredients.length; i++) {
        const ingredients = recipe.ingredients[i];

        const isIngredientMatch = regex.test(ingredients.ingredient);

        if (isIngredientMatch) {
          list.push(recipe);
        }
      }
      if (isNameMatch) {
        list.push(recipe);
      }
      if (isDescriptionMatch) {
        list.push(recipe);
      }
    }
  });
};
```

(Annexe 2 : Test jsBench.ch)

The screenshot shows the jsBench.ch website interface. The top navigation bar includes the JSBEN.CH logo, a 'Run' button, a prompt to 'Please login/register to save & publish tests', and links for 'Sponsor', 'Settings', and 'Sign in'. The main content area is divided into two panels. The left panel displays the test status as 'finished' with a performance result of '467467.4 ops/s ± 0.67%' and a note 'Fastest'. The right panel shows the JavaScript code for the benchmark, which includes a search algorithm and a list of recipes. The code is as follows:

```
const searchAlgo = (recipes, input) => {
  const list = [];
  for (let i = 0; i < recipes.length; i++) {
    const recipe = recipes[i];
    const recipeName = recipe.name.toLowerCase();
    const recipeDescription = recipe.description.toLowerCase();
    const regex = new RegExp("(?:" + input.toLowerCase() + ")", "i");
    const isNameMatch = regex.test(recipeName);
    const isDescriptionMatch = regex.test(recipeDescription);

    for (let i = 0; i < recipe.ingredients.length; i++) {
      const ingredients = recipe.ingredients[i];

      const isIngredientMatch = regex.test(ingredients.ingredient);

      if (isIngredientMatch) {
        list.push(recipe);
      }
    }
    if (isNameMatch) {
      list.push(recipe);
    }
    if (isDescriptionMatch) {
      list.push(recipe);
    }
  }
}

const searchAlgo = (recipes, input) => {
  return recipes.filter(recipe => {
    const list = [];
    for (let i = 0; i < recipes.length; i++) {
      const recipe = recipes[i];
      const recipeName = recipe.name.toLowerCase();
      const recipeDescription = recipe.description.toLowerCase();
      const regex = new RegExp("(?:" + input.toLowerCase() + ")", "i");
      const isNameMatch = regex.test(recipeName);
      const isDescriptionMatch = regex.test(recipeDescription);

      for (let i = 0; i < recipe.ingredients.length; i++) {
        const ingredients = recipe.ingredients[i];

        const isIngredientMatch = regex.test(ingredients.ingredient);

        if (isIngredientMatch) {
          list.push(recipe);
        }
      }
      if (isNameMatch) {
        list.push(recipe);
      }
      if (isDescriptionMatch) {
        list.push(recipe);
      }
    }
  });
};

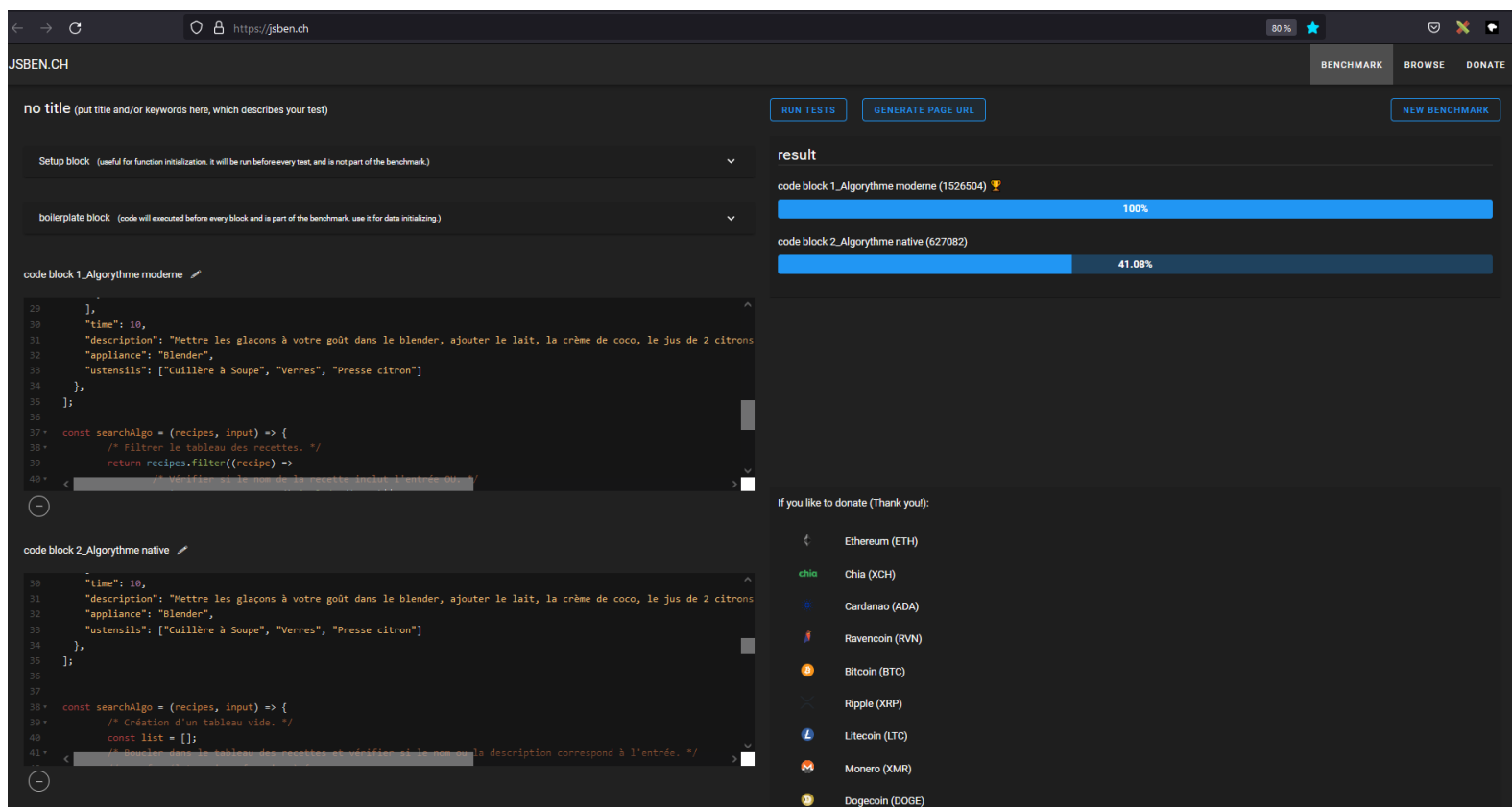
const searchAlgo = (recipes, input) => {
  return recipes.filter(recipe => {
    const list = [];
    for (let i = 0; i < recipes.length; i++) {
      const recipe = recipes[i];
      const recipeName = recipe.name.toLowerCase();
      const recipeDescription = recipe.description.toLowerCase();
      const regex = new RegExp("(?:" + input.toLowerCase() + ")", "i");
      const isNameMatch = regex.test(recipeName);
      const isDescriptionMatch = regex.test(recipeDescription);

      for (let i = 0; i < recipe.ingredients.length; i++) {
        const ingredients = recipe.ingredients[i];

        const isIngredientMatch = regex.test(ingredients.ingredient);

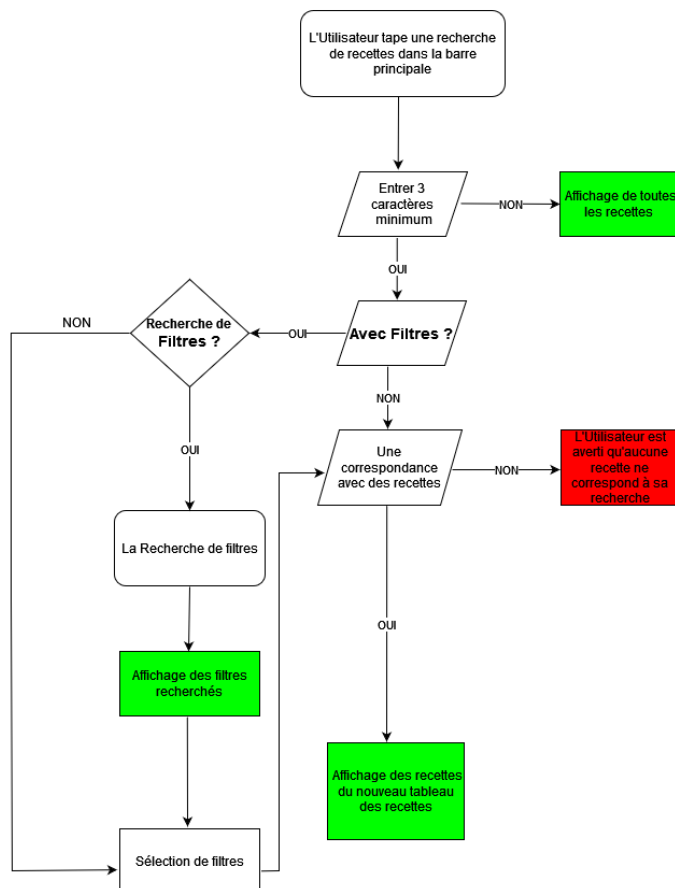
        if (isIngredientMatch) {
          list.push(recipe);
        }
      }
      if (isNameMatch) {
        list.push(recipe);
      }
      if (isDescriptionMatch) {
        list.push(recipe);
      }
    }
  });
};
```

(Annexe 2 : Test jsBench.ch)



Diagramme

Annexe : La Fiche d'investigation de fonctionnalité de la recherche générale



Utilisation de la méthode filter pour créer un nouveau tableau si la valeur de la recherche correspond à une valeur liée aux ingrédients, aux ustensils, aux appareils d'une recette