Les petits plats

Fiche d’investigation fonctionnalité

Fonctionnalité : Filtrer les recettes dans l’interface utilisateur
Problématique : Réaliser une recherche rapide et performante dans les titres, les descriptions et les ingrédients des recettes. L'utilisateur doit pouvoir filtrer les recettes selon deux axes : une barre principale pour rechercher des mots ou des groupes de lettres dans le titre, les ingrédients ou la description et une recherche par mots-clés dans les ingrédients, les ustensiles ou l'appareil.

Implémentation	
Utilisation de méthodes de tableau natives	Ne pas utiliser les méthodes de tableau natives (uniquement les boucles)
Scénario 1 1. L'algorithme de recherche vérifie si des caractères sont entrés dans la chaîne de recherche et si leur nombre est supérieur à 2. 2. L'utilisateur n'a pas saisi de caractères ou le nombre de caractères n'est pas supérieur à 2. 3. Aucun mot-clé n'a été sélectionné 4. Retournez le résultat de la recherche	
Scénario 2 1. L'algorithme de recherche vérifie si des caractères sont entrés dans la chaîne de recherche et si leur nombre est supérieur à 2.	

2. L'utilisateur a saisi des caractères et le nombre de caractères est supérieur à 2.
3. Aucune correspondance de recherche
4. Retour "Aucun résultat"

Scénario 3

1. L'algorithme de recherche vérifie si des caractères sont entrés dans la chaîne de recherche et si leur nombre est supérieur à 2.
2. L'utilisateur a saisi des caractères et le nombre de caractères est supérieur à 2.
3. Il y a des correspondances de recherche
4. Aucun mot-clé n'a été sélectionné
5. Renvoyer les résultats de la recherche

Scénario 4

1. L'algorithme de recherche vérifie si des caractères sont entrés dans la chaîne de recherche et si leur nombre est supérieur à 2.
2. L'utilisateur n'a pas saisi de caractères et le nombre de caractères n'est pas supérieur à 2.
3. Les mots-clés ont été sélectionnés
4. Retourner les résultats de la recherche

Avantages	Inconvénients	Avantages	Inconvénients
1. Le temps d'exécution est 10 fois plus rapide en moyenne 2. Le code est plus lisible 3. Plus facile à maintenir 4. Plus facile à mettre en œuvre	1. Il faut connaître la syntaxe actuelle.	-	1. Le temps d'exécution est en moyenne 10 fois plus lent 2. Le code est devenu moins lisible 3. Plus difficile à entretenir 4. Plus difficile à mettre en œuvre

Le code avec utilisation de méthodes de tableau natives

```
const search = (badgesList) => {
    console.time( label: 'search')
    const filteredRecipesBySearchInput = recipes.filter(({name,ingredients,description}) => JSON.stringify( value: {name,ingredients,description}).toLowerCase().includes(searchValue))
    // Filter recipes by badges
    const filterRecipes = filteredRecipesBySearchInput.filter(({ingredients, appliance, ustensils}) => {
        // If badgesList is empty, return all recipes
        if (!badgesList.length) {
            return true
        }
        // If any of the badges match any of the recipes, return true
        return badgesList.every(({name, type}) => {
            if (type === 'ingredients') {
                return ingredients.some(({ingredient}) => ingredient.toLowerCase() === name.toLowerCase())
            } else if (type === 'appliances') {
                return appliance.toLowerCase() === name.toLowerCase()
            } else if (type === 'utensils') {
                return ustensils.some((utensil) => utensil.toLowerCase() === name.toLowerCase())
            }
        })
    })
    console.timeEnd( label: 'search')
    render(filterRecipes, badgesList)
}
```

Le code sans utilisation de méthodes de tableau natives

```
const searchWithCycles = (badgesList) => {
    console.time( label: 'searchWithCycles')
    const filteredRecipesBySearchInput = []
    const filterRecipes = []
    let flag = false
    for (let i = 0; i < recipes.length; i++) {
        const regex = new RegExp(searchValue,  flags: "g");
        const {name,ingredients,description} = recipes[i]
        if (JSON.stringify( value: {name,ingredients,description}).toLowerCase().match(regex)) {
            filteredRecipesBySearchInput.push(recipes[i])
        }
    }
    if (badgesList.length){
        for (let i = 0; i < badgesList.length; i++) {
            const {name, type} = badgesList[i]
            for (let j = 0; j < filteredRecipesBySearchInput.length; j++) {
                flag = someElementHasValue(name, type, filteredRecipesBySearchInput[j])
                if (flag) {
                    filterRecipes.push(filteredRecipesBySearchInput[j])
                }
            }
        }
    }else {
        filterRecipes.push(...filteredRecipesBySearchInput)
    }
    console.timeEnd( label: 'searchWithCycles')
    if (badgesList.length < 2) {
        render(filterRecipes, badgesList)
    } else {
        render(getDuplicateElements(filterRecipes), badgesList)
    }
}
```

```
function someElementHasValue(name, type, element) {
    let flag = false
    const {ingredients, appliance, ustensils} = element
    if (type === 'ingredients') {
        for (let i = 0; i < ingredients.length; i++) {
            let condition = ingredients[i].ingredient.toLowerCase() === name.toLowerCase()
            if (condition) {
                flag = true
                break
            }
        }
    } else if (type === 'appliances') {
        if (appliance === name) {
            flag = true
        }
    } else if (type === 'utensils') {
        for (let i = 0; i < ustensils.length; i++) {
            let condition = ustensils[i].toLowerCase() === name.toLowerCase()
            if (condition) {
                flag = true
                break
            }
        }
    }
    return flag
}

function getDuplicateElements(array) {
    const result = []
    for (let i = 0; i <= array.length; i++) {
        for (let j = 0; j <= array.length; j++) {
            if (i !== j && array[i] === array[j] && result.indexOf(array[i]) === -1) {
                result.push(array[i]);
                break;
            }
        }
    }
    return result
}
```

Conclusion

A mon avis, l'utilisation de méthodes natives du langage Javascript facilite les étapes de développement, de maintenance du code, et gagne souvent en rapidité du script.