Les petits plats

Fiche d’investigation de fonctionnalité

Fonctionnalité : Rechercher dans l’input principal	**Fonctionnalité #2**
Problématique : Accéder **rapidement** à une recette correspondant à un besoin de l’utilisateur (depuis le fichier de recettes déjà reçu) .	

Option 1 : Approche code fonctionnel

Dans cette option, nous avons un filtre de recette qui utilise le code fonctionnel et renvoie un tableau avec les recettes correspondant à la recherche.

La fonction analyse le tableau par un map, elle filtre (.filter) le tableau de recettes sous condition de la présence (.includes) de la valeur de l’input et renvoie un tableau de recettes filtrées
On utilise les méthodes .map, .filter, .includes, .some pour filtrer un tableau en fonction d’une chaîne de minimum 3 caractères dans les propriétés description, name et ingredients

Avantages	Inconvénients
⊕ la fonction tient en peu de ligne (6 lignes) ⊕ la lisibilité et la maintenance sont facilités: avec les méthodes, on comprends ce que l’on fait à la lecture ⊕ la fonction est plus rapide, mais de manière insignificative (20660 op/sec)	⊖ Les méthodes font parties des nouvelles pratiques et peuvent ne pas être familière de tous les développeurs

Conclusion:

Cette approche est plus rapide et plus maniable

Option 2: Approche boucles natives

Dans cette option, on utilise des boucles native « for » pour l’itération du fichier recette, renvoie un tableau avec les recettes correspondant à la recherche.

La fonction loop, va analyser chaque itérance du tableau de recettes, elle vérifie la présence de la valeur taper dans chaque itérance, si la valeur est présente elle rajoute la recette à un tableau qu’elle renvoie.

On utilise les boucles for et indexOf pour retrouver la présence d’une chaîne de minimum 3 caractères dans une chaîne plus longue (les propriétés description, name et ingredients)

Avantages	Inconvénients
⊕ Plus facile d’écrire une boucle dans une boucle ⊕ Présent depuis l’origine du langage, et dans plusieurs langage, on peut construire beaucoup d’algorithme avec for	⊖ Plus de lignes à écrire (19 lignes) ⊖ Lisibilité complexe du code ⊖ Equivalent en rapidité, voire légèrement moins performant.

CONCLUSION

Cette approche est fait appel à du code natif, plus difficile à aborder en lecture rapide.

Solution retenue :

J’ai retenu la solution de code fonctionnel pour ses avantages de clarté et de simplicité du code. On note un gain de performance faible avec une différence de moins de 1% , mais un code plus facilement maintenable utilisant les avantages des méthodes de javascript.

Annexe 1:

Screenshot du code, extrait provenant de la page filters.js branche master et filters.js branche NativeLoops, visuel vsCode.

```
// renvoie une liste de recettes filtrée avec e de l'event listener, et en array: Recipes
export function filterThroughMainInput (e, array) {
    const filteredArrayDescription = array.filter(recipe => refit(recipe.description).includes(refit(e.target.value)))
    const filteredArrayName = array.filter(recipe => refit(recipe.name).includes(refit(e.target.value)))
    const filteredArrayIngredients = array.filter(recipe => recipe.ingredients.some(ing => refit(ing.ingredient).includes(refit(e.target.value))))
    const setOfMainSearchInput = [...new Set([...filteredArrayDescription, ...filteredArrayName, ...filteredArrayIngredients])]
    // console.log("ensemble des results de mainInput : ", setOfMainSearchInput)
    return setOfMainSearchInput
}
```

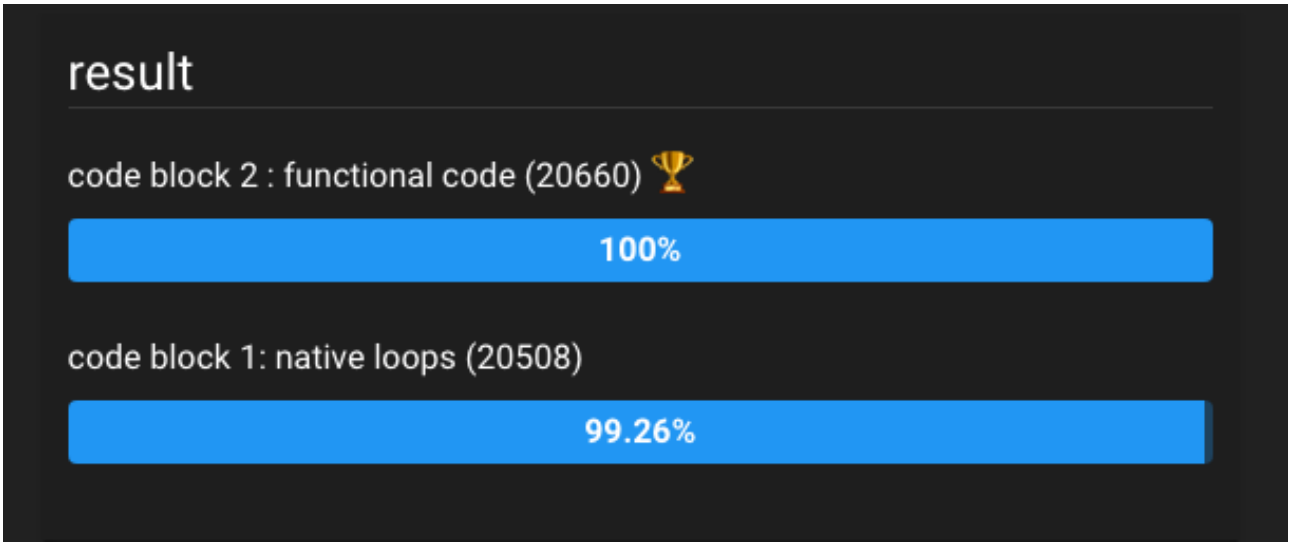
```
export function loopThroughMainInput(e, array){
    let mixedShortenedArray=[]
    let searchedDescription, searchedName, searchedIngredient
    for (let i = 0; i < array.length; i++) {
        searchedDescription = refit(array[i].description).indexOf(refit(e.target.value))
        searchedName = refit(array[i].name).indexOf(refit(e.target.value))
        if (searchedDescription!== -1 && mixedShortenedArray.indexOf(array[i])== -1) {
            mixedShortenedArray.push(array[i])
        } if (searchedName!== -1 && mixedShortenedArray.indexOf(array[i])== -1) {
            mixedShortenedArray.push(array[i])
        }
        for (let j = 0; j < array[i].ingredients.length; j++){
            searchedIngredient = refit(array[i].ingredients[j].ingredient).indexOf(refit(e.target.value))
            if (searchedIngredient !== -1 && mixedShortenedArray.indexOf(array[i]) == -1){
                mixedShortenedArray.push(array[i])
            }
        }
    }
    //console.log("mixed test", mixedShortenedArray);
    return mixedShortenedArray
}
```

Annexe 2 :

Résultat de l'analyse de performance provenant de jsben.ch, screenshot.

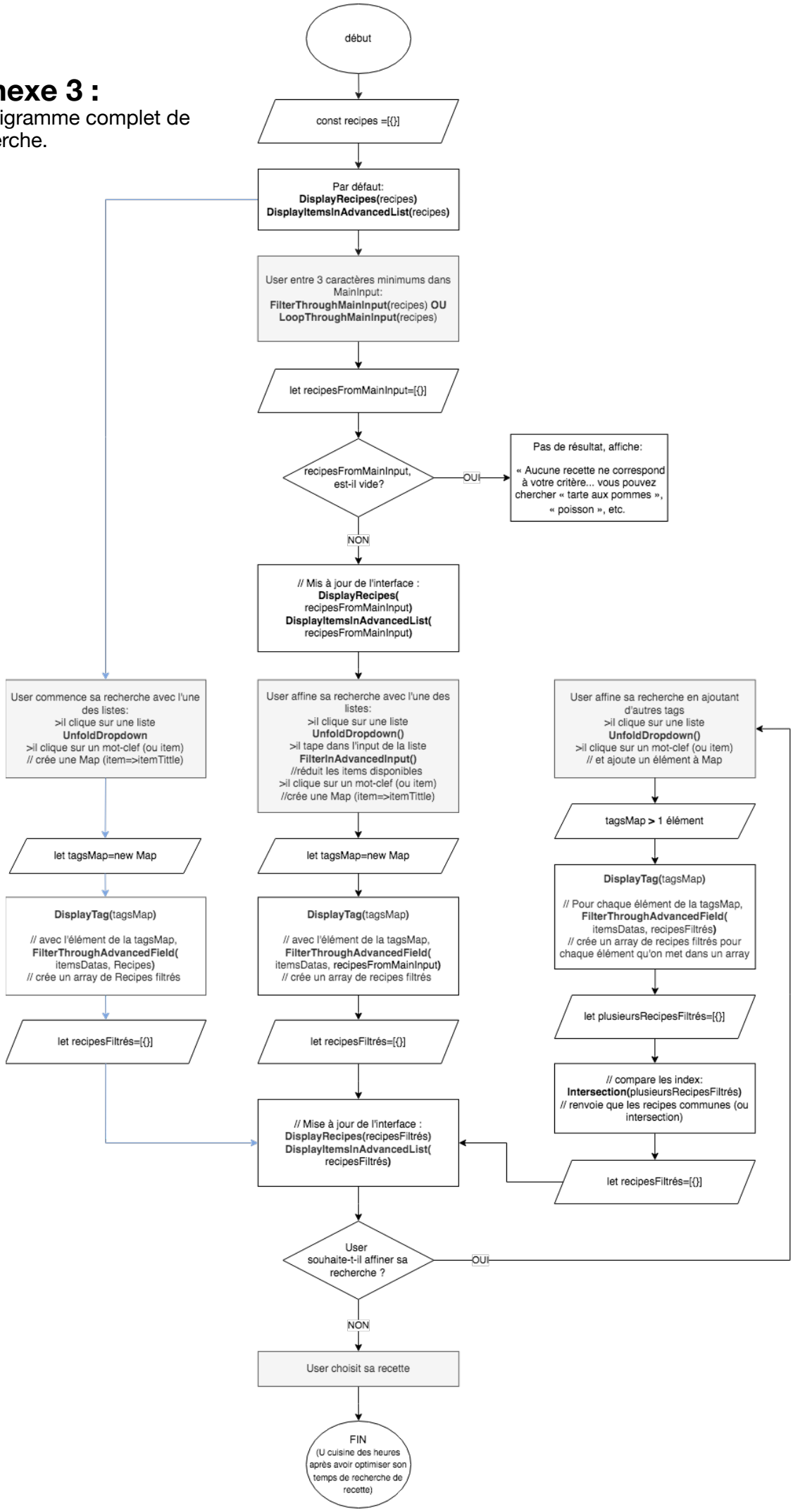
Lien vers le test: https://jsben.ch/gUW10

Block 1= LoopThroughMainInput
Block 2= FilterThroughMainInput



Annexe 3 :

Algorigramme complet de recherche.



Annexe 4:

Algorigrammes des fonctions de filtres de la recherche générale