Les petits plats

Fiche d'investigation de fonctionnalité

Fonctionnalité	Moteur de recherche
Problématique	Nous cherchons à concevoir un algorithme permettant à aux utilisateurs de chercher une recette en rentrant des mots clés dans la barre de recherche. La recherche doit être le plus fluide possible et donner l'impression de recherche instantanée. Nous commencerons la recherche après 3 caractères tapés.

Algorithmes

Nous avons construit 6 algorithmes différents, chaque algorithme suit la même logique de construction, la seule différence est la méthode utilisée pour filtrer les recettes.

Algorithme A - Programmation fonctionnelle (forEach)	
Avantages : • Facile à lire et à maintenir • Sur 20 recherches, il a eu 1 fois le meilleur score	**Inconvénients :** • Temps de réponse moyen le plus lent • Sur 20 recherches, il eu 5 fois le moins bon score
Moyenne sur 20 recherches : 0.3038ms	
Médiane sur 20 recherches : 0.3097ms	

Algorithme B - Programmation fonctionnelle (map)	
Avantages : ● Facile à lire et à maintenir ● Sur 20 recherches, il a eu 2 fois le meilleur score	**Inconvénients :** ● Sur 20 recherches, il eu 5 fois le moins bon score
Moyenne sur 20 recherches : 0.2981ms	
Médiane sur 20 recherches : 0.3020ms	

Algorithme C - Programmation fonctionnelle (reduce)	
Avantages : ● Facile à lire et à maintenir ● Sur 20 recherches, il a eu 9 fois le meilleur score ● Sur 20 recherches, il n'a jamais eu le moins bon score ● Temps de réponse moyen le plus rapide ● Temps de réponse médian le plus rapide	**Inconvénients :** ● Un peu moins facile à lire que les algorithmes A & B
Moyenne sur 20 recherches : 0.2851ms	
Médiane sur 20 recherches : 0.2955ms	

Algorithme D - Programmation fonctionnelle (filter)	
Avantages : ● Facile à lire et à maintenir ● Sur 20 recherches, il a eu 1 fois le meilleur score	**Inconvénients :** ● Sur 20 recherches, il eu 2 fois le moins bon score
Moyenne sur 20 recherches : 0.2908ms	
Médiane sur 20 recherches : 0.3040ms	

Algorithme E - Boucle native (for)	
Avantages : ● Sur 20 recherches, il a eu 4 fois le meilleur score	**Inconvénients :** ● Plus difficile à lire et à maintenir que les méthodes fonctionnelles ● Sur 20 recherches, il eu 4 fois le moins bon score ● temps de réponse médian le plus lent
Moyenne sur 20 recherches : 0.3021ms	
Médiane sur 20 recherches : 0.3104ms	

Algorithme F - Boucle native (white)	
Avantages : ● Sur 20 recherches, il a eu 3 fois le meilleur score	**Inconvénients :** ● Plus difficile à lire et à maintenir que les méthodes fonctionnelles ● Sur 20 recherches, il eu 4 fois le moins bon score
Moyenne sur 20 recherches : 0.3021ms	
Médiane sur 20 recherches : 0.3104ms	

Solution retenue

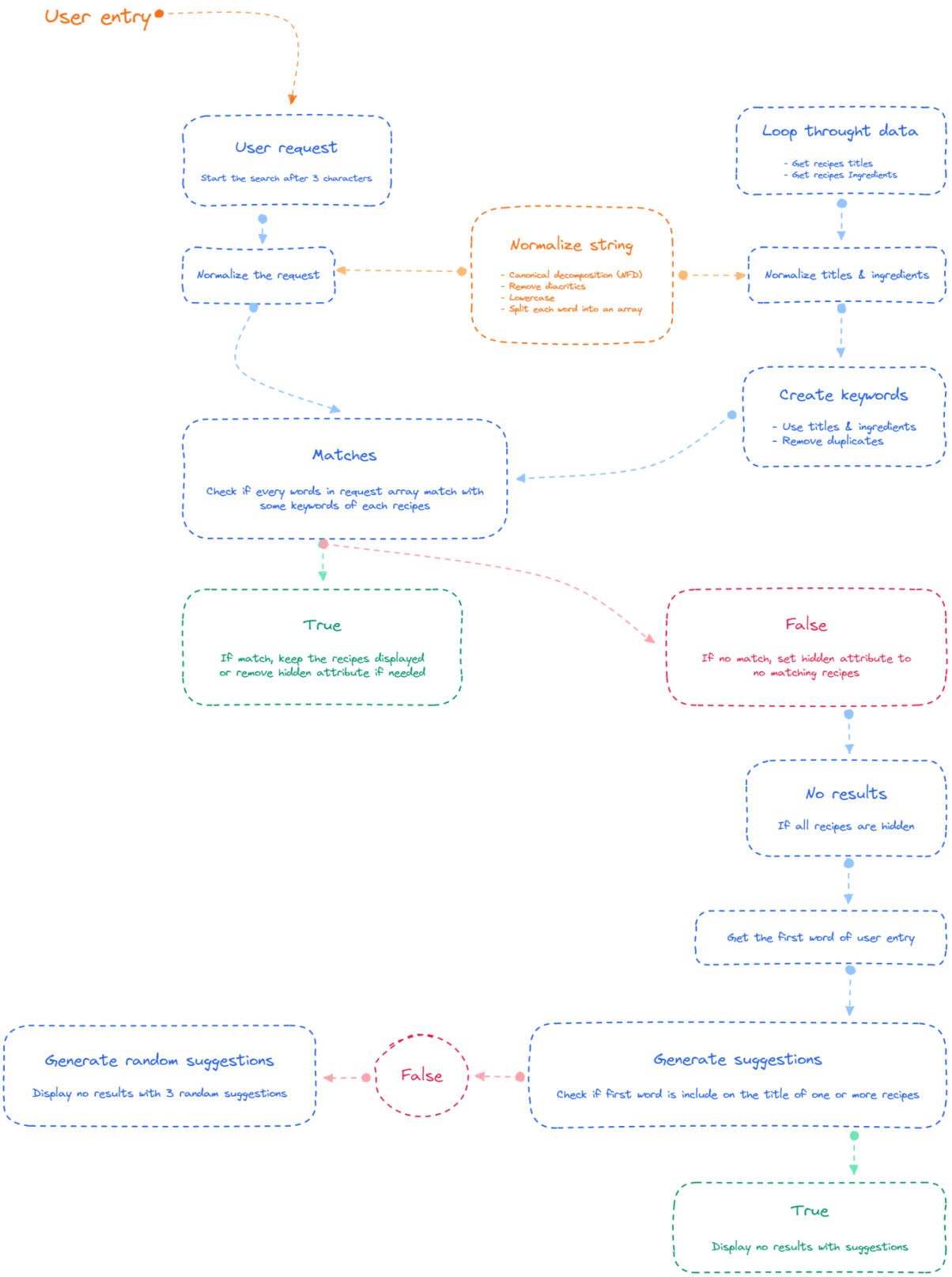
De manière générale, toutes les méthodes ont des résultats très proches avec des différences de performances au millième de millisecondes ce qui n'est pas assez significatif pour poser une conclusion.

Cependant après tous les tests nous pouvons remarquer que l'algorithme C s'en sort globalement mieux que les autres et à l'avantage d'être plus lisible et maintenable que les boucles natives (for / while). C'est pourquoi nous opterons pour celui-ci.

Il faut quand même garder à l'esprit que notre base de données est très restreinte (uniquement 50 recettes), il est donc difficile de savoir si ce dernier s'en sortira tout aussi bien avec un nombre beaucoup plus important de recettes.

Annexes

Annexe #1 : Logique partagé par tous les algorithmes

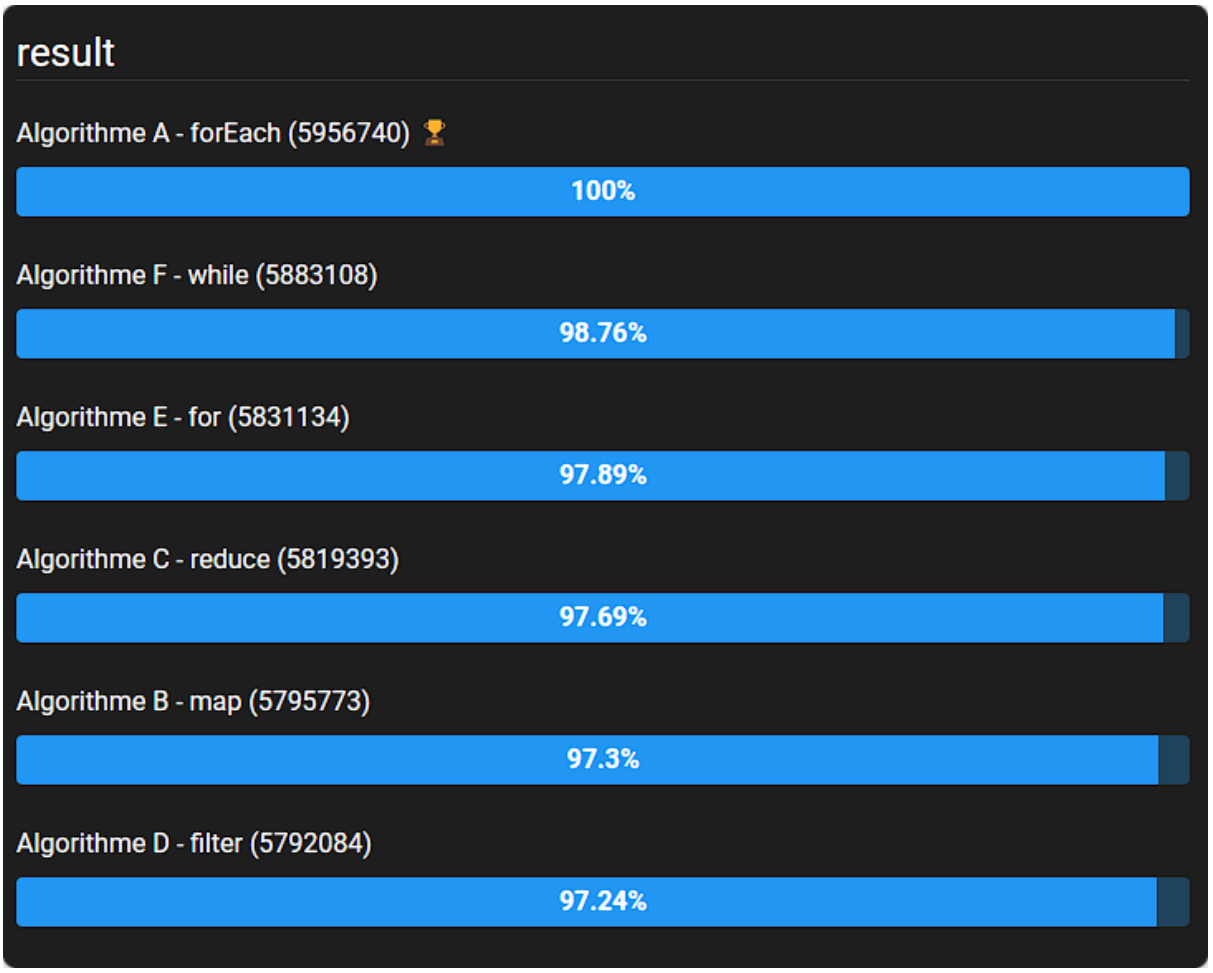


Annexe #2 : Benchmark personnalisé pour les différents algorithmes

	Algo A	Algo B	Algo C	Algo D	Algo E	Algo F
Limonade	0,2917	0,2930	0,2690	0,2771	0,2888	0,2839
Poisson	0,3203	0,3010	0,2629	0,2629	0,3728	0,3081
Poulet	0,3560	0,3049	0,3230	0,3030	0,3142	0,2939
Tarte	0,3091	0,2952	0,3000	0,3040	0,3040	0,3442
Salade	0,3030	0,2930	0,2839	0,3162	0,3140	0,3000
Shake	0,3669	0,3379	0,3508	0,3040	0,2961	0,3069
Pates	0,2690	0,3379	0,2839	0,3230	0,3279	0,3071
Fondant au chocolat	0,3159	0,3560	0,3142	0,2988	0,3027	0,2969
Spaghettis à la bolognaise	0,3093	0,3250	0,2781	0,3118	0,3921	0,3220
quiche lorraine	0,3022	0,3132	0,2751	0,3218	0,3440	0,3027
Lait	0,3149	0,3140	0,3010	0,3311	0,3149	0,2930
Oeuf	0,3398	0,2720	0,3281	0,2888	0,3079	0,3040
Farine	0,3018	0,2969	0,2920	0,2849	0,3020	0,3328
Mousse au citron	0,3176	0,3208	0,3022	0,3120	0,3130	0,3711
Pizza	0,3398	0,2939	0,2922	0,2991	0,3359	0,3391
Pruneaux	0,3081	0,3030	0,2961	0,3108	0,2859	0,3650
Lasagne courgettes et chèvre	0,3220	0,2920	0,3101	0,3010	0,2590	0,2913
Purée de carottes	0,3020	0,2966	0,2949	0,3079	0,3262	0,3110
noresults	0,3101	0,3220	0,3000	0,3118	0,2930	0,2981
Escape key	0,0759	0,0930	0,0442	0,0471	0,0469	0,0432
Average	0,3038	0,2981	0,2851	0,2908	0,3021	0,3007
Median	0,3097	0,3020	0,2955	0,3040	0,3104	0,3054

Les résultats sont exprimés en millisecondes avec un arrondi supérieur à 3 décimales. Les cellules vertes représentent les temps les plus rapides, les cellules rouges les temps les plus lents.

Annexe #3 : Benchmark JSBENC.ch pour les différents algorithmes



Je ne sais pas sur quel environnement JSBENCH.ch effectue ses tests mais dans notre cas les résultats me semblent vraiment incohérents, de plus ils sont toujours différents. C'est pourquoi j'ai préféré faire les tests moi même. Mais pour respecter toutes les contraintes du projet je mets tout de même en annexe les résultats mais il est fortement conseillé de se référer à l'annexe #2.

URL : https://jsben.ch/e8WrA

Annexe #4 : Snippet algorithme A

Search Algo A

```
if (event.target.value.length >= 3) {
  // Get search value
  const searchTerms = Normalize(event.target.value)

  recipeData.forEach((recipe, index) => {
    // Get recipe title and ingredients
    const recipeTitle = Normalize(recipe.name)
    const recipeIngredients = Normalize(recipe.ingredients.join(' '))

    // Get keywords
    const keywords = [...new Set([...recipeTitle, ...recipeIngredients])]

    // Check if search terms are in keywords
    const isMatch = searchTerms.every(term => keywords.some(keyword =>
keyword.includes(term)))

    // Hide or show recipe card
    recipeCards[index].hidden = !isMatch
  })
} else {
  recipeCards.forEach(recipeCard => (recipeCard.hidden = false))
}
```

Annexe #5 : Snippet algorithme B

Search Algo B

```
if (event.target.value.length >= 3) {
  // Get search value
  const searchTerms = Normalize(event.target.value)

  // Get keywords
  const keywords = recipeData.map(recipe => {
    // Get recipe title and ingredients
    const recipeTitle = Normalize(recipe.name)
    const recipeIngredients = Normalize(recipe.ingredients.join(' '))

    // Get keywords
    return [...new Set([...recipeTitle, ...recipeIngredients])]
  })

  // Check if search terms are in keywords
  const isMatch = keywords.map(keyword =>
    searchTerms.every(term => keyword.some(keyword => keyword.includes(term)))
  )

  // Hide or show recipe cards
  isMatch.forEach((match, index) => (recipeCards[index].hidden = !match))
} else {
  recipeCards.forEach(recipeCard => (recipeCard.hidden = false))
}
```

Annexe #6 : Snippet algorithme C

Search Algo C

```
if (event.target.value.length >= 3) {
  // Get search value
  const searchTerms = Normalize(event.target.value)

  // Get keywords
  const keywords = recipeData.reduce((acc, recipe) => {
    // Get recipe title and ingredients
    const recipeTitle = Normalize(recipe.name)
    const recipeIngredients = Normalize(recipe.ingredients.join(' '))

    // Get keywords
    acc.push([...new Set([...recipeTitle, ...recipeIngredients])])

    return acc
  }, [])

  // Check if search terms are in keywords
  const isMatch = keywords.reduce((acc, keyword) => {
    acc.push(searchTerms.every(term => keyword.some(keyword => keyword.includes(term))))

    return acc
  }, [])

  // Hide or show recipe cards
  isMatch.forEach((match, index) => (recipeCards[index].hidden = !match))
} else {
  recipeCards.forEach(recipeCard => (recipeCard.hidden = false))
}
```

Annexe #7 : Snippet algorithme D

Search Algo D

```
if (event.target.value.length >= 3) {
  // Get search value
  const searchTerms = Normalize(event.target.value)

  // Get keywords
  const keywords = recipeData
    .map(recipe => {
      // Get recipe title and ingredients
      const recipeTitle = Normalize(recipe.name)
      const recipeIngredients = Normalize(recipe.ingredients.join(' '))

      // Get keywords
      return [...new Set([...recipeTitle, ...recipeIngredients])]
    })
    .filter(keyword => searchTerms.every(term => keyword.some(keyword =>
keyword.includes(term))))

  // Hide or show recipe cards
  recipeCards.forEach((recipeCard, index) => (recipeCard.hidden = !keywords[index]))
} else {
  recipeCards.forEach(recipeCard => (recipeCard.hidden = false))
}
```

Annexe #8 : Snippet algorithme E

Search Algo E

```
if (event.target.value.length >= 3) {
  // Get search value
  const searchTerms = Normalize(event.target.value)

  // Get keywords
  const keywords = []
  for (let i = 0; i < recipeData.length; i++) {
    // Get recipe title and ingredients
    const recipeTitle = Normalize(recipeData[i].name)
    const recipeIngredients = Normalize(recipeData[i].ingredients.join(' '))

    // Get keywords
    keywords.push([...new Set([...recipeTitle, ...recipeIngredients])])
  }

  // Check if search terms are in keywords
  const isMatch = []
  for (let i = 0; i < keywords.length; i++) {
    isMatch.push(searchTerms.every(term => keywords[i].some(keyword =>
keyword.includes(term))))
  }

  // Hide or show recipe cards
  for (let i = 0; i < isMatch.length; i++) {
    recipeCards[i].hidden = !isMatch[i]
  }
} else {
  recipeCards.forEach(recipeCard => (recipeCard.hidden = false))
}
```

Annexe #9 : Snippet algorithme F

Search Algo F

```
if (event.target.value.length >= 3) {
  // Get search value
  const searchTerms = Normalize(event.target.value)

  // Get keywords
  const keywords = []
  let i = 0
  while (i < recipeData.length) {
    // Get recipe title and ingredients
    const recipeTitle = Normalize(recipeData[i].name)
    const recipeIngredients = Normalize(recipeData[i].ingredients.join(' '))

    // Get keywords
    keywords.push([...new Set([...recipeTitle, ...recipeIngredients])])

    i++
  }

  // Check if search terms are in keywords
  const isMatch = []
  i = 0
  while (i < keywords.length) {
    isMatch.push(searchTerms.every(term => keywords[i].some(keyword =>
keyword.includes(term))))

    i++
  }

  // Hide or show recipe cards
  i = 0
  while (i < isMatch.length) {
    recipeCards[i].hidden = !isMatch[i]

    i++
  }
} else {
  recipeCards.forEach(recipeCard => (recipeCard.hidden = false))
}
```