**Fiche d'investigation fonctionnalité**

**Fonctionnalité :** Recherche principale - Site web “Les Petits Plats”

**Problématique :**

Afin d’offrir une expérience utilisateur optimale, nous cherchons à implémenter une fonctionnalité de recherche efficace et performante pour le site “Les Petits Plats”.

**Option 1 : Recherche par boucles natives JavaScript**

Dans cette option, nous utilisons des boucles JavaScript (for, while) pour parcourir le tableau de recettes et effectuer la recherche.

Avantages

* Contrôle précis sur l’itération des données.
* Possibilité d’arrêter la boucle dès qu’une correspondance est trouvée (potentiellement plus rapide pour de petits ensembles de données).
* Facilité de compréhension pour les développeurs débutants.

Inconvénients

* Moins performant pour de grandes quantités de données.
* Code potentiellement plus verbeux et moins lisible.
* Risque plus élevé d’erreurs de logique dans les boucles imbriquées.

JavaScript

function searchRecipes(query, recipes) {

const results = [];

for (let i = 0; i < recipes.length; i++) {

const recipe = recipes[i];

if (

recipe.name.toLowerCase().includes(query.toLowerCase()) ||

recipe.description.toLowerCase().includes(query.toLowerCase()) ||

recipe.ingredients.some(ing => ing.ingredient.toLowerCase().includes(query.toLowerCase()))

) {

results.push(recipe);

}

}

return results;

}

Complexité algorithmique (Big O)

* Temps : O(n \* m), où n est le nombre de recettes et m est le nombre moyen d’ingrédients par recette.
* Espace : O(k), où k est le nombre de recettes correspondant aux critères de recherche.

Explication : Dans le pire des cas, nous devons parcourir toutes les recettes (n) et pour chaque recette, nous pourrions avoir à parcourir tous ses ingrédients (m). La méthode includes() a une complexité de O(p) où p est la longueur de la chaîne, mais nous pouvons la considérer comme constante pour cette analyse.

**Option 2 : Recherche utilisant les méthodes d’array (filter)**

Cette option utilise les méthodes d’array modernes de JavaScript, notamment filter, pour effectuer la recherche.

Avantages

* Code plus concis et lisible.
* Approche fonctionnelle qui réduit les effets de bord.
* Potentiellement plus rapide pour de grandes quantités de données (optimisations internes du navigateur).
* Facilité d’ajout de critères de recherche supplémentaires.

Inconvénients

* Moins intuitif pour les développeurs débutants.
* Parcourt toujours l’ensemble des données (ne peut pas s’arrêter prématurément).
* Potentiellement moins performant pour de très petits ensembles de données.

JavaScript

function searchRecipes(query, recipes) {

return recipes.filter(recipe =>

recipe.name.toLowerCase().includes(query.toLowerCase()) ||

recipe.description.toLowerCase().includes(query.toLowerCase()) ||

recipe.ingredients.some(ing => ing.ingredient.toLowerCase().includes(query.toLowerCase()))

);

}

Complexité algorithmique (Big O)

* Temps : O(n \* m), où n est le nombre de recettes et m est le nombre moyen d’ingrédients par recette.
* Espace : O(k), où k est le nombre de recettes correspondant aux critères de recherche.

Explication : Bien que nous utilisions des méthodes d’array plus modernes, la complexité temporelle reste la même que pour l’approche avec des boucles natives. Nous devons toujours parcourir toutes les recettes (n) et pour chaque recette, nous examinons potentiellement tous ses ingrédients (m). La méthode filter() crée un nouveau tableau, ce qui explique la complexité spatiale.

**Solution retenue**

Nous avons retenu l’approche utilisant les méthodes d’array (Option 2) pour plusieurs raisons :

1. Le code est plus lisible et maintenable, ce qui facilitera les futures évolutions de la fonctionnalité.
2. Cette approche est généralement plus performante pour de grandes quantités de données, ce qui est crucial pour un site de recettes qui pourrait potentiellement contenir des milliers d’entrées.
3. L’utilisation de méthodes d’array comme filter et some rend le code plus déclaratif, permettant de comprendre rapidement l’intention de la recherche.
4. Cette méthode s’adapte plus facilement à l’ajout de nouveaux critères de recherche ou à la modification des critères existants.

Bien que la complexité algorithmique soit identique pour les deux approches, les optimisations internes des navigateurs pour ces méthodes d’array peuvent offrir de meilleures performances dans la pratique, en particulier pour de grands ensembles de données.

Considérations supplémentaires sur la complexité

Pour améliorer davantage les performances, surtout pour de grands ensembles de données, on pourrait envisager des structures de données plus avancées (comme des index inversés) ou des algorithmes de recherche plus sophistiqués, qui pourraient potentiellement réduire la complexité à O(log n) ou mieux.

Résultats des tests de performance

1. For Loop Search :
   * Temps d’exécution : 1.15087890625 ms
   * Description : Cette méthode utilise des boucles for pour parcourir chaque recette et chaque terme de recherche. Bien que les boucles for soient généralement performantes, elles peuvent être moins optimisées pour certaines opérations de tableau en JavaScript.
2. Array Methods Search :
   * Temps d’exécution : 0.68603515625 ms
   * Description : Cette méthode utilise des méthodes de tableau comme filter et every. Ces méthodes sont souvent optimisées en interne par le moteur JavaScript, ce qui peut les rendre plus rapides pour certaines opérations, surtout lorsqu’il s’agit de manipuler des tableaux.

Pourquoi les méthodes de tableau sont plus rapides :

* Optimisation interne : Les méthodes de tableau comme filter, map, et reduce sont souvent optimisées par les moteurs JavaScript modernes (comme V8 pour Chrome et Edge). Cela signifie qu’elles peuvent être plus rapides pour certaines opérations par rapport aux boucles for.
* Moins de code : Les méthodes de tableau permettent d’écrire moins de code, ce qui peut réduire le temps d’exécution global en minimisant les opérations redondantes.
* Lisibilité et maintenabilité : Bien que cela n’affecte pas directement les performances, le code utilisant les méthodes de tableau est souvent plus lisible et plus facile à maintenir, ce qui peut indirectement améliorer les performances en facilitant les optimisations futures.

**Conclusion**

Les résultats montrent que l’utilisation des méthodes de tableau peut offrir des performances supérieures pour certaines opérations de recherche par rapport aux boucles for. Cependant, il est toujours important de tester les performances dans le contexte spécifique de votre application, car les résultats peuvent varier en fonction des données et des opérations spécifiques.

Annexe :

