Fiche d'investigation de fonctionnalité

Fonctionnalité : Recherche par mots clé en 2 étapes	**Fonctionnalité #2**
Problématique : Obtenir les meilleures performances possibles pour une recherche par mot clé, implémentée sur un champ de recherche principal, puis sur les résultats de cette recherche, par cat®gorie (recherche avancée)	

Option 1 : Utilisation d'un algorithme de recherche lin®aire
Recherche principale : La recherche s'accomplit en passant par chacune des recettes, en comparant le mot cl® avec le contenu de chaque recette, dans l'ordre o½elles sont r®cup®r®es de l'api.
Recherche avanc®e : s'ex®cute sur les r®sultats de la 1¯re recherche, mais en cherchant uniquement dans la cat®gorie d'o½vient le mot.

Avantages	**Inconvénients**
Data structures basiques (implémentation plus facile)	Moins bonnes performances

Option 2 : Utilisation d'un algorithme de recherche non lin®aire ('Trie tree')
Recherche principale : A réception des données api, chacune des recettes est mappée dans un arbre de tri. La recherche s'effectue dans cet arbre.

Recherche avancée : si la recherche principale est active, un nouvel arbre est construit à partir des résultats, et la recherche avancée s'effectue dans ce nouvel arbre

Avantages	**Inconvénients**
Meilleures performances si la taille des données croit	Au 1er chargement de la page la construction du tree prend quelques millisecondes (il est ensuite socké dans le local storage)

Nombre de characteres minimum à entrer dans le champ principal : 3

Nombre de characteres minimum à entrer dans le champ secondaire (catégories) : 3

COMPARAISONS CHIFFREES DES PERFORMANCES (utilisant l'api 'performance.now()')

		LINEAR SEARCH	TRIE SEARCH
recherche du mot 'citron': temps de récupération des résultats	FIREFOX	9 milliseconds	1 millisecond
	CHROME	6 milliseconds	0.19 millisecond
recherche du mot 'coco': temps de récupération des résultats	FIREFOX	10 milliseconds	1 millisecond
	CHROME	6.5 milliseconds	0.30 millisecond
recherche du mot 'rhubarbe': temps de récupération des résultats	FIREFOX	6 milliseconds	1 millisecond
	CHROME	4.6 milliseconds	0.10 millisecond

CONCLUSION
Les performances de recherche d'un même mot par les 2 algorithmes présentent des écarts importants, sur un set de données pourtant de petite taille. La recherche non linéaire est clairement la plus performante.

Pour compléter l'analyse, il serait malgré tout nécessaire d'évaluer le temps de construction initiale de l'arbre.

Solution retenue :
Recherche non linéaire par 'tree trie' (ou 'prefix tree).

Fiche d'investigation de fonctionnalité

Fonctionnalité : Recherche par mots clé en 2 étapes	**Fonctionnalité #2**
Problématique : Obtenir les meilleures performances possibles pour une recherche par mot clé, implémentée sur un champ de recherche principal, puis sur les résultats de cette recherche, par cat®gorie (recherche avancée)	

Option 1 : DETAILS DE L' Utilisation d'un algorithme de recherche non lin®aire : PREFIX TRIE

Recherche principale : La recherche s'accomplit en passant par chacune des recettes, en comparant le mot cl® avec le contenu de chaque recette, dans l'ordre o½elles sont r®cup®r®es de l'api.

DESCRIPTION DU 'TIME COMPLEXITY' de la recherche non linéaire
(uniquement la recherche dans le trie, omettant la construction préalable)

FUNCTION	BIG O 'n' representant la taille des nodes (qui est function du nbre de mots entrés dans le trie)
⇒ CHERCHER UN MOT dans le trie	
► Split mot to array	➡ O(n)
► For Loop : chaque lettre du mot	➡ O(n)
▷ **node has** letter ?	➡ O(1)
▷ **node get** letter	➡ O(1)
► Go to last node (recursion)	
▷ node keys **For Loop**	➡ O(n)
▷ suggestions array **includes** recipe ?	➡ O(n)
▷ suggestions **push** recipe	➡ O(1)
⇒ TOTAL	⇒ 4 opérations de O(n) ⇒ + 3 opérations de O(1) ⇢ **O(n)**

Conclusion : la recherche non linéaire se traduit bien par une complexité de O(n).
Pour une évaluation plus précise, il faudrait évaluer également les performances liées à la construction initiale du trie.

Fiche d'investigation de fonctionnalité

Fonctionnalité : Recherche par mots clé en 2 étapes	**Fonctionnalité #2**
Problématique : Obtenir les meilleures performances possibles pour une recherche par mot clé, implémentée sur un champ de recherche principal, puis sur les résultats de cette recherche, par cat®gorie (recherche avancée)	

Option 1 : DETAILS DE L' Utilisation d'un algorithme de recherche lin®aire

Recherche principale : La recherche s'accomplit en passant par chacune des recettes, en comparant le mot cl® avec le contenu de chaque recette, dans l'ordre o½elles sont r®cup®r®es de l'api.

DESCRIPTION DU 'TIME COMPLEXITY' de la recherche linéaire

FUNCTION	BIG O 'n' representant la taille des données entrantes
⇒ POUR CHAQUE RECETTE	➡ O(n)
► Chercher dans le nom	
▷ **split** name to array	➡ O(n)
▷ array **filter**(word)	➡ O(n)
▷ word **includes**(term)	➡ O(n)
► Chercher dans la description	
▷ **split** desc to array	➡ O(n)
▷ array **filter()** word	➡ O(n)
▷ 2 * regex **Test**()	➡ 2 * O(n)
▷ word.**includes**(term)	➡ O(n)
► Chercher dans les ingredients	
▷ **For each** ingredient of array	➡ O(n)
▷ name **toLowerCase()**	➡ O(n)
▷ name **includes**(term)	➡ O(n)
⇒ TOTAL	⇒ O(n) * 11 opérations de O(n) => **O(n)2**

Conclusion : la recherche linéaire devrait se traduire par une complexité de O(n), non O(n2) (qui en fait une compléxité de type quadratique). Pour cela, le code doit être optimisé pour simplifier le nombre d'opérations.