Fiche d’investigation de fonctionnalité

Fonctionnalité :

recherche principale des recettes selon le terme saisi par l’utilisateur

Problématique :

Afin de répondre aux attentes des utilisateurs, nous cherchons à avoir une séquence de recherche la plus rapide possible.

Architecture :

La recherche & le filtrage de recettes sont effectués, en parcourant chaque recettes sur le tableau des recettes.

Puis, les listes des champs de recherche avancés sont mises à jour en fonction des nouvelles recettes filtrées.

La 1ère approche, avec **l'itérateur "for"** utilise la **fonction "searchRecipes".**

La seconde, avec la **méthode itérative "reduce"**, utilise la **fonction "filterByReduce".**

On retrouve sur ces 2 fonctions:

- un **paramètre d'entrée** "***searchForm***", un objet représentant les requêtes de recherche, et contenant:
  - un terme (terme de recherche saisi par l'utilisateur)
  - un tableau d'ingrédients liés à la recherche
  - un tableau d'appareils liés à la recherche
  - un tableau d'ustensiles liés à la recherche

- le **paramètre de sortie,** un tableau contenant les recettes (correspondant aux critères de recherche)

- A chaque itération, la fonction vérifie si la recette correspond aux critères de recherche:
  - si le nom, la description ou l'un des ingrédients contient-il le terme recherché
  - si l'appareil de la recette est présent dans la liste des appareils, ou si la liste des appareils est vide
  - si tous les ustensiles spécifiés sont présents dans la liste des ustensiles de la recette
  - si tous les ingrédients spécifiés sont présents dans la liste des ingrédients de la recette.

- **Si** une **recette répond à tous ces critères**, elle est **ajoutée au tableau** (tableau vide créé ou accumulateur (acc) selon l'approche).

- **MAJ des champs de recherche avancés:** la fonction utilise les résultats de la recherche pour mettre à jour les éléments des 3 champs de recherche avancés (ingrédients,

appareils & ustensiles), ainsi les filtres affichent uniquement les éléments contenus dans les nouvelles recettes.

- **Les 2 approches utilisent une fonction permettant de rechercher efficacement les recettes en fonction de plusieurs critères, et de mettre à jour les éléments des champs de recherche avancés.**

Algorithme - Option 1 - Boucles natives avec l'instruction "for"

- Code à retrouver sur la branche "***native-loops***" du repository Github (voir fonction "***searchRecipes***", fonction de recherche globale, incluant la fonctionnalité de recherche principale avec l'instruction "***for***")

- A propos de cette instruction "***for***"
  - permet de créer une boucle composée de ***3 expressions*** (optionnelles):
    - ***initialisation*** *(souvent en déclarant une variable représentant un compteur...)*
    - ***condition*** *(une expression évaluée avant chaque itération de la boucle, si elle est vérifiée l'instruction est exécutée...)*
    - ***expression finale*** *(évaluée à la fin de chaque itération, et généralement utilisée pour mettre à jour ou incrémenter le compteur (ou variable d'initialisation)...)*
  - ces 3 expressions sont suivies par ***l'instruction:***
    - exécutée tant que la condition de la boucle est vérifiée
  - on peut utilise les instructions comme "***break***" (pour stopper & sortir de la boucle en cours) ou encore "***continue***" (pour arrêter l'exécution des instructions et passer directement à l'itération suivante).

Avantages:

- fonctionnement simple et explicite
- facilite la compréhension du code

Inconvénients:

- la gestion de l'index (tel un compteur) peut rendre le code difficile à lire (notamment sur les tableaux multi-dimensionnels)
- un code plus long (en comparaison avec celui utilisant la méthode itérative "***reduce***")

Algorithme - Option 2 - Méthode "reduce" issue de l'objet Array

- **Code à retrouver** sur la branche ***"main" (***voir fonction "***filterByReduce***", fonction de recherche globale, incluant la fonctionnalité de recherche principale)
- A propos de la méthode ***reduce*** ::
  - "*... applique une fonction qui est un « accumulateur » et qui traite chaque valeur d'une liste (de la gauche vers la droite) afin de la réduire à une seule valeur*".
  - utilise la **notion d'accumulateur:** "*Un accumulateur est une variable qui va nous permettre de passer le résultat de la 1ère itération en paramètre de la seconde itération et ainsi de suite...*"
- Méthode réputée pour sa puissance, et sa polyvalence mais parfois complexe à utiliser.

Avantages:

- code plus concis, plus court (moins de ligne de code qu'avec l'itérateur "**for**")
- code plus simple à lire et à comprendre
- code moins sujet aux bugs, méthode `**reduce**` respecte le principe d'immutabilité (travail directement sur un nouveau tableau sans modifier le tableau d'origine...)

Inconvénients:

- demande de comprendre le fonctionnement de cette méthode, et de connaître certaines notions, comme celle d'accumulateur...
- son utilisation demande de la vigilance pour éviter des erreurs (comme par exemple, ne pas oublier de retourner l'accumulateur (arr)...).

Performances:

- on remarque que les 2 algorithmes ont chacun des performances variables selon les cas d'utilisation. Dans le cas de la fonctionnalité de la recherche principale, l'approche la plus performante est celle de la méthode **reduce**.

Solution retenue :

Après avoir testé les performances des 2 algorithmes sur la fonctionnalité de recherche principale, l'approche utilisant la méthode "***reduce***" est plus performant que l'approche utilisant l'instruction "**for**".

Nous avons donc retenu l'approche avec la méthode ***"reduce"***, pour des raisons de performances, de lisibilité et de qualité de code.

LIEN VERS REPO.:

- **Code final contenant la solution retenue**, avec la méthode "**reduce**" (utilisée dans la fonction de recherche "**filterByReduce**"), solution à retrouver sur :
  - branche "array-object"
  - branche principale "main" (code refactorisé)
- **Code contenant la solution avec l'itérateur "for"** (utilisée dans la fonction de recherche "**searchRecipes**"), solution à retrouver sur :
  - branche "native-loops"
- Application en déploiement continue (depuis la branche "main" du repo. Github)

LIEN VERS COMPARAISON PERFORMANCES:

- https://jsben.ch/bh4vt

LIEN VERS W3C HTML/CSS VALIDATOR:

- validation code HTML
- validation code CSS

LIEN VERS DOCUMENTATIONS:

- doc. MDN itérateur "for"
- doc. MDN méthode "reduce"