Documentation

Structure des Composants

- **RecipeForm** : Ce composant est responsable de la création et de l'édition des recettes grâce à plusieurs formulaires (titre, temps de cuisson, ingrédients et méthode). Il s'occupe de la vérification des champs et appelle les fonctions du parent (`App.jsx`) pour ajouter ou modifier des recettes.
- **RecipeCard** : Ce composant affiche les différentes informations d'une recette dans une carte. Il inclut deux boutons pour modifier et supprimer une recette, ces derniers font également appel aux fonctions de `App.jsx`.
- **App** : Il s'agit du composant principal. Il gère les différents composants, transmet les informations entre eux et leur donne accès à certaines fonctions comme la suppression, la modification et la création de recettes.

Fonctionnalités Implémentées

1. **Ajout de Recettes** : Les utilisateurs peuvent ajouter de nouvelles recettes via un formulaire.
2. **Modification de Recettes** : Les utilisateurs peuvent modifier une recette existante en cliquant sur le bouton "Modifier" d'une carte.
3. **Suppression de Recettes** : Les utilisateurs peuvent supprimer une recette via le bouton "Supprimer" sur une carte.
4. **Persistance des Données** : Les recettes sont stockées localement dans IndexedDB, permettant de conserver les données même après un rechargement de la page.

Explication du code/fonction

recipeForm

- **handleChange** : Cette fonction met à jour l'état local du formulaire à chaque saisie utilisateur. Elle utilise l'identifiant du champ pour mettre à jour la valeur correspondante dans l'objet `form`.
- **handleSubmit** : Cette fonction vérifie que tous les champs du formulaire sont valides avant d'appeler soit `handleAdd` (pour ajouter une recette) soit `handleEdit` (pour modifier une recette). Une fois l'action effectuée, elle réinitialise le formulaire.

recipeCard

- **Affichage des ingrédients** : Les ingrédients sont affichés sous forme de liste. Si les ingrédients ne sont pas déjà un tableau, ils sont convertis en tableau en les séparant par des virgules.
- **Boutons Modifier/Supprimer** : Ces boutons appellent les fonctions passées en props (`handleEdit` et `handleDelete`) pour effectuer les actions correspondantes.

App

- **chargerRecettes** : Cette fonction récupère toutes les recettes stockées dans la base de données Dexie et met à jour l'état `recipeList` pour les afficher dans l'application.

- **sauvegarderRecette** : Cette fonction ajoute une nouvelle recette dans la base de données Dexie et recharge la liste des recettes.
- **modifierForm** : Cette fonction met à jour une recette existante dans la base de données Dexie. Si l'ID de la recette est invalide ou introuvable, elle affiche une erreur dans la console.
- **handleDelete** : Cette fonction supprime une recette de la base de données Dexie et met à jour la liste des recettes affichées.

Dexie.js

- Dexie.js est utilisé pour gérer une base de données IndexedDB. Les recettes sont définies avec les champs `id`, `title`, `cuissonTime`, `ingredients`, et `methode`. L'ajout, la suppression et la mise à jour des recettes sont asynchrones et gérées avec des promesses.

Difficultés Rencontrées

Fonction d'action sur les objets

Notre première difficulté a été de donner aux différents composants un **accès aux objets** recette. On a finalement décidé de gérer cela grâce à des fonctions **"callback"** que l'on a définies dans App.jsx et données aux composants enfants pour qu'ils puissent librement les utiliser.

Gestion des Données avec Dexie.js

Nous avons également eu du mal au début pour l'**import** des données au **chargement** de la page. Nous avons rencontré certains problèmes comme le fait d'avoir plusieurs fois l'import depuis Dexie ou encore la création infinie de nouvelles recettes pour l'initialisation.

Fonctions asynchrones

Nous nous sommes également battus avec certaines fonctions pour des raisons d'**asynchronie**. Nous avions des fonctions qui n'attendaient pas le retour de celles appelées précédemment, ce qui posait des problèmes. Dans un premier temps, nous avions essayé de résoudre ce problème avec des `async/await`, mais finalement nous avons trouvé plus simple et lisible d'utiliser des `.then` et des `.catch`.