D'accord ! Je vais expliquer chaque étape de manière détaillée, en abordant les notions principales et en expliquant le code étape par étape pour chaque fichier de votre projet. Cela inclut les composants React, Redux, ainsi que les concepts clés que vous devez comprendre.

1. `Accueil.js` (Composant Principal pour Ajouter des Transactions)

Importation des dépendances

```
import React, { useState } from "react";
import { useDispatch, useSelector } from "react-redux";
import {
  ajouterTransaction,
  retirerTransaction,
  enregistrerTransactions,
} from "../redux/action";
```

- `useState` : C'est un hook de React qui permet de gérer l'état local dans un composant fonctionnel. Il est utilisé ici pour gérer les valeurs des champs comme le montant et l'opération.
- `useDispatch` : C'est un hook fourni par Redux qui vous permet d'envoyer des actions (comme `ajouterTransaction` ou `retirerTransaction`) pour modifier l'état global.
- `useSelector` : Ce hook vous permet de lire des valeurs depuis le store Redux. Par exemple, ici, on lit le `compteur` et les `transactions temporaires` du store Redux.

Initialisation des états

```
const [nombre, setNombre] = useState("");
const [operation, setOperation] = useState("+");
const compteur = useSelector((state) => state.compteur);
const temporaire = useSelector((state) => state.temporaire);
const dispatch = useDispatch();
```

- `nombre` : Gère le montant que l'utilisateur saisit.
- `operation` : Gère l'opération sélectionnée, soit `+` pour ajouter, soit `-` pour retirer.
- `compteur` : La valeur du solde actuel (provenant de Redux).
- `temporaire` : Les transactions temporaires avant qu'elles ne soient enregistrées (provenant de Redux).
- `dispatch` : Permet d'envoyer des actions Redux pour mettre à jour l'état global.

Fonction de calcul (`handleCalculer`)

```
const handleCalculer = () => {
  if (nombre) {
    const montant = parseFloat(nombre);
    if (operation === "+") {
      dispatch(ajouterTransaction(montant));
    } else {
```

```
        dispatch(retirerTransaction(montant));
      }
      setNombre("");
    } else {
      alert("Veuillez saisir un montant valide !");
    }
  };
```

- `parseFloat(nombre)` : Convertit la chaîne de caractères `nombre` en un nombre décimal.
- `dispatch(ajouterTransaction(montant))` : Si l'opération est `+`, l'action `ajouterTransaction` est envoyée avec le montant, ce qui met à jour l'état global via Redux.
- `dispatch(retirerTransaction(montant))` : Si l'opération est `-`, l'action `retirerTransaction` est envoyée avec le montant.
- **Réinitialisation de `nombre`** : Après avoir effectué le calcul, le champ de saisie est réinitialisé.

Fonction d'enregistrement (`handleEnregistrer`)

```
const handleEnregistrer = () => {
  dispatch(enregistrerTransactions());
  alert("Transactions enregistrées avec succès !");
};
```

- `dispatch(enregistrerTransactions())` : Cette action enregistre les transactions temporaires dans la liste des transactions enregistrées en envoyant une action à Redux.
- **Affichage de l'alerte** : Après l'enregistrement, une alerte apparaît pour informer l'utilisateur que les transactions ont été enregistrées.

Rendu du composant

```
return (
  <div>
    <h1 className="text-center mb-4">Gestion d'Argent de Poche</h1>
    <div className="card p-4 shadow">
      <div className="mb-3">
        <label className="form-label">Montant :</label>
        <input
          type="number"
          className="form-control"
          value={nombre}
          onChange={(e) => setNombre(e.target.value)}
        />
      </div>
      <div className="mb-3">
        <label className="form-label me-2">Opération :</label>
        <div className="form-check form-check-inline">
          <input
            className="form-check-input"
```

```
            type="radio"
            value="+"
            checked={operation === "+"}
            onChange={() => setOperation("+")}
          />
          <label className="form-check-label">Ajouter</label>
        </div>
        <div className="form-check form-check-inline">
          <input
            className="form-check-input"
            type="radio"
            value="-"
            checked={operation === "-"}
            onChange={() => setOperation("-")}
          />
          <label className="form-check-label">Retirer</label>
        </div>
      </div>
      <div className="d-grid gap-2">
        <button onClick={handleCalculer} className="btn btn-primary">
          Calculer
        </button>
        <button onClick={handleEnregistrer} className="btn btn-success">
          Enregistrer
        </button>
      </div>
    </div>
    <h2 className="text-center mt-4">Solde : {compteur} DH</h2>
    <h3 className="mt-4">Transactions temporaires :</h3>
    <ul>
      {temporaire.map((transaction, index) => (
        <li key={index}>
          {transaction.type === "+" ? "Ajouté" : "Retiré"} :{" "}
          {Math.abs(transaction.montant)} MAD, Solde après opération :{" "}
          {transaction.compteur} MAD
        </li>
      ))}
    </ul>
  </div>
);
```

- Affiche un formulaire pour entrer le montant et choisir l'opération (`Ajouter` ou `Retirer`).
- Les transactions temporaires sont affichées sous forme de liste.
- Le solde est mis à jour en temps réel.

2. `App.js` (Routage de l'application)

```
import React from "react";
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import Accueil from "./accueil"; // Le composant d'accueil pour ajouter des
transactions
```

```
import Liste from "./liste"; // Le composant pour afficher l'historique des
transactions
import Navbar from "./navbar"; // Importation de la barre de navigation

function App() {
  return (
    <Router>
      {/* Ajout de la barre de navigation */}
      <Navbar />

      <Routes>
        <Route path="/" element={<Accueil />} />{" "}
        {/* Route vers la page d'accueil */}
        <Route path="/liste" element={<Liste />} /> {/* Route vers la liste des
transactions */}
      </Routes>
    </Router>
  );
}

export default App;
```

- `<Router>` : Permet de gérer le routage dans l'application. Il englobe toute l'application pour activer la navigation entre les composants.
- `<Routes>` : Contient toutes les routes de l'application.
- `<Route path="/" element={<Accueil />} />` : Cette route affichera le composant `Accueil` lorsque l'utilisateur se rend à la racine (`/`).
- `<Route path="/liste" element={<Liste />} />` : Affiche le composant `Liste` lorsque l'utilisateur accède à `/liste`.

3. `Liste.js` (Affichage des Transactions Enregistrées)

```
import React from "react";
import { useSelector } from "react-redux";

function Liste() {
  const transactions = useSelector((state) => state.transactions);

  return (
    <div>
      <h2 className="text-center mb-4">Liste des Transactions</h2>
      <div className="table-responsive">
        <table className="table table-bordered table-striped">
          <thead className="table-dark">
            <tr>
              <th>Operation</th>
              <th>Montant</th>
              <th>Compteur</th>
            </tr>
          </thead>
```

```
            <tbody>
              {transactions.map((transaction, index) => (
                <tr key={index}>
                  <td>{transaction.type === "+" ? "Ajouté" : "Retiré"}</td>
                  <td>{Math.abs(transaction.montant)} MAD</td>
                  <td>{transaction.compteur} MAD</td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      </div>
    );
}

export default Liste;
```

- `useSelector` : Permet de lire l'état global de Redux, ici la liste des `transactions`.
- Affiche une table avec les transactions enregistrées, chaque ligne montrant l'opération (ajoutée ou retirée), le montant, et le solde après l'opération.

4. `Reducer.js` (Gestion de l'État Redux)

```
import { AJOUTER, RETIRER, ENREGISTRER } from "./types";

const initialState = {
  transactions: [], // Transactions enregistrées
  temporaire: [], // Transactions temporaires
  compteur: 0, // Solde actuel
};

const reducer = (state = initialState, action) => {
  switch (action.type) {
    case AJOUTER:
      return {
        ...state,
        temporaire: [
          ...state.temporaire,

          {
            type: "+",
            montant: action.montant,
            compteur: state.compteur + action.montant,
          },
        ],
        compteur: state.compteur + action.montant,
      };

    case RETIRER:
      return {
        ...state,
```

```
        temporaire: [
          ...state.temporaire,
          {
            type: "-",
            montant: action.montant,
            compteur: state.compteur - action.montant,
          },
        ],
        compteur: state.compteur - action.montant,
      };

    case ENREGISTRER:
      return {
        ...state,
        transactions: [...state.transactions, ...state.temporaire],
        temporaire: [],
      };

    default:
      return state;
  }
};

export default reducer;
```

- `AJOUTER` : Ajoute un montant au solde et enregistre la transaction temporairement.
- `RETIRER` : Retire un montant du solde et enregistre la transaction temporairement.
- `ENREGISTRER` : Transfère les transactions temporaires vers les transactions enregistrées et réinitialise la liste des transactions temporaires.

5. Actions Redux (`action.js`)

```
import { AJOUTER, RETIRER, ENREGISTRER } from "./types";

export const ajouterTransaction = (montant) => ({
  type: AJOUTER,
  montant,
});

export const retirerTransaction = (montant) => ({
  type: RETIRER,
  montant,
});

export const enregistrerTransactions = () => ({
  type: ENREGISTRER,
});
```

- `ajouterTransaction` : Action pour ajouter un montant au solde.
- `retirerTransaction` : Action pour retirer un montant du solde.

- `enregistrerTransactions` : Action pour enregistrer les transactions temporaires.

Conclusion

Avec ce projet, vous comprenez comment utiliser Redux pour gérer l'état global d'une application React. Les actions et les réducteurs sont essentiels pour effectuer des mises à jour de l'état global (ajouter, retirer des montants). Chaque composant est responsable d'afficher et d'interagir avec l'état, tout en utilisant des hooks comme `useDispatch` et `useSelector` pour communiquer avec Redux.