DEVOWFS 202 CONTROLE 2 EXPLIQUATION

Accueil.js

Voici une explication détaillée de votre composant **Accueil.js** pour mieux comprendre son fonctionnement.

Fonctionnalité du Composant

Le composant `Accueil` permet à l'utilisateur de :

1. **Ajouter des informations** : Le nom de la transaction, le montant, et le type (ajouté ou retiré).
2. **Enregistrer une transaction** : Ajouter les informations saisies à une liste de transactions via la fonction `saveTransaction` passée en prop.
3. **Afficher le solde total** : Montrer le solde total disponible (prop `solde`) en temps réel.

Structure et Fonctionnement

1. Utilisation des États Locaux

Le composant utilise trois états pour gérer les données saisies :

- `nom` : Stocke le nom de la transaction saisie par l'utilisateur.

```
const [nom, setNom] = useState("");
```

- `montant` : Stocke le montant saisi.

```
const [montant, setMontant] = useState("");
```

- `type` : Stocke le type de transaction (`+` pour ajout ou `-` pour retrait).

```
const [type, setType] = useState("+");
```

Ces états sont mis à jour via les gestionnaires d'événements des champs de saisie.

---

2. Props Reçues

Le composant reçoit deux props :

1. `solde` : Le solde total actuel (affiché dynamiquement).
2. `saveTransaction` : Une fonction de rappel (callback) passée par le parent pour enregistrer une transaction.

3. Fonction `handleSave`

La fonction principale `handleSave` est déclenchée lorsqu'on clique sur le bouton **Save** :

```
const handleSave = () => {
  if (nom && montant) {
    saveTransaction(nom, montant, type); // Enregistre la transaction.
    setNom(""); // Réinitialise le champ nom.
    setMontant(""); // Réinitialise le champ montant.
  } else {
    alert("Veuillez remplir tous les champs !");
  }
};
```

Étapes :

1. Vérifie si les champs `nom` et `montant` sont remplis.
2. Si oui :
   - Appelle la fonction `saveTransaction` avec les valeurs actuelles des états (`nom`, `montant`, et `type`).
   - Réinitialise les champs après l'enregistrement.
3. Sinon :
   - Affiche un message d'erreur pour demander à l'utilisateur de remplir tous les champs.

---

4. Affichage du Formulaire

Nom

```
<input
  type="text"
  className="form-control"
  value={nom}
  onChange={(e) => setNom(e.target.value)}
/>
```

- **Type** : Champ texte.
- `value` : Lié à l'état `nom`.
- `onChange` : Met à jour l'état `nom` à chaque saisie de l'utilisateur.

Montant

```
<input
  type="number"
  className="form-control"
```

```
  value={montant}
  onChange={(e) => setMontant(e.target.value)}
/>
```

- **Type** : Champ numérique.
- Fonctionne de la même manière que le champ **Nom**, mais pour l'état `montant`.

Type

```
<input
  className="form-check-input"
  type="radio"
  value="+"
  checked={type === "+"}
  onChange={() => setType("+")}
/>
```

- Boutons radio pour choisir le type de transaction.
- Utilise l'état `type` pour déterminer quel bouton est sélectionné (`checked`).

---

5. Solde Total

Le solde total est affiché en bas du formulaire :

```
<h5>
  Solde Total: <span className="text-success">{solde} MAD</span>
</h5>
```

- La prop `solde` est affichée directement.
- Le style `text-success` (classe Bootstrap) colore le texte en vert.

---

6. Bouton Save

```
<button className="btn btn-success me-2" onClick={handleSave}>
  Save
</button>
```

- **Classe Bootstrap** :
  - `btn btn-success` : Style vert pour signaler une action positive.
  - `me-2` : Ajoute une marge à droite pour espacer les boutons (si vous en ajoutez d'autres).
- **`onClick`** : Appelle `handleSave` pour gérer l'enregistrement de la transaction.

Points Clés

Logique

1. **Validation des Champs** :
   - Vérifie que le nom et le montant sont remplis avant d'enregistrer.
2. **Réinitialisation** :
   - Après l'enregistrement, les champs de saisie sont vidés pour permettre une nouvelle transaction.
3. **Propagation des Données** :
   - Les données saisies sont envoyées au composant parent via `saveTransaction`.

Design

- Utilise **Bootstrap** pour un design simple et propre :
  - `form-control` : Style standard pour les champs de saisie.
  - `form-check` : Style pour les boutons radio.
  - `btn btn-success` : Style pour le bouton Save.

Récapitulatif des États

État	Rôle	Modifié par	Utilisé par
`nom`	Stocke le nom de la transaction.	Saisie dans le champ texte.	Lors de l'enregistrement.
`montant`	Stocke le montant de la transaction.	Saisie dans le champ numérique.	Lors de l'enregistrement.
`type`	Stocke le type de transaction (`+` ou `-`).	Boutons radio.	Lors de l'enregistrement.

Améliorations Possibles

1. **Ajout d'un champ de date** :
   - Pour enregistrer la date de chaque transaction.
2. **Validation plus stricte** :
   - Vérifier que le montant est un nombre positif.
3. **Messages d'erreur** :
   - Afficher des erreurs plus détaillées si les champs sont mal remplis.

Liste.js

Voici une explication détaillée de votre composant **Liste.js** pour mieux comprendre son fonctionnement.

---

1. Fonctionnalité Principale

Le composant `Liste` sert à afficher la liste des transactions enregistrées sous forme d'un tableau. Chaque transaction contient trois informations :

- **Nom** : Nom associé à la transaction.
- **Montant** : Montant de la transaction (valeur absolue affichée).
- **Type** : Type de transaction (`+` pour ajouté et `-` pour retiré).

---

2. Structure du Composant

Props Reçues

Le composant reçoit une seule prop :

- `transactions` : Un tableau d'objets, où chaque objet représente une transaction avec les propriétés suivantes :
  - `nom` : Le nom de la transaction.
  - `montant` : Le montant de la transaction (peut être positif ou négatif).
  - `type` : Le type de la transaction (`+` ou `-`).

Exemple de structure de la prop `transactions` :

```
[
  { nom: "Achat livre", montant: -200, type: "-" },
  { nom: "Salaire", montant: 3000, type: "+" }
]
```

Structure HTML

Le tableau HTML affiche les informations sous forme de lignes (`<tr>`) :

1. **En-tête du tableau (`<thead>`)** : Contient les titres des colonnes (`Nom`, `Montant`, et `Type`).
2. **Corps du tableau (`<tbody>`)** : Génère dynamiquement une ligne pour chaque transaction.

Exemple : Résultat Attendu

Si `transactions` contient :

```
[
  { nom: "Achat livre", montant: -200, type: "-" },
  { nom: "Salaire", montant: 3000, type: "+" }
]
```

Le tableau affichera :

Nom	Montant	Type

Nom	Montant	Type
Achat livre	200 MAD	Retiré
Salaire	3000 MAD	Ajouté

---

3. Explications du Code

Structure du JSX

1. **Titre Principal**

```
<h2 className="text-center mb-4">Liste des Transactions</h2>
```

   - Affiche un titre centré.
   - `mb-4` (classe Bootstrap) ajoute une marge en bas pour espacer le titre du tableau.

2. **Conteneur du Tableau**

```
<div className="table-responsive">
```

   - Permet de rendre le tableau défilable horizontalement sur les petits écrans (responsive design).

3. **Tableau**

```
<table className="table table-bordered table-striped">
```

   - `table` : Classe Bootstrap pour activer le style par défaut des tableaux.
   - `table-bordered` : Ajoute des bordures aux cellules du tableau.
   - `table-striped` : Alterne les couleurs des lignes pour une meilleure lisibilité.

4. **En-tête du Tableau**

```
<thead className="table-dark">
```

   - La classe `table-dark` de Bootstrap applique un fond sombre pour les en-têtes.

5. **Corps du Tableau**

```
<tbody>
  {transactions.map((transaction, index) => (
    <tr key={index}>
      <td>{transaction.nom}</td>
```

```
        <td>{Math.abs(transaction.montant)} MAD</td>
        <td>{transaction.type === "+" ? "Ajouté" : "Retiré"}</td>
      </tr>
    ))}
  </tbody>
```

- `transactions.map` : Parcourt le tableau `transactions` pour créer une ligne (`<tr>`) par transaction.
- **Clé unique (`key`)** : Chaque ligne est identifiée par un index unique pour optimiser le rendu.
- **Contenu des Colonnes (`<td>`)** :
  - **Nom** : Affiche directement la propriété `nom` de la transaction.
  - **Montant** : Affiche la valeur absolue de `montant` suivie de "MAD".
  - **Type** : Vérifie si le type est `+` (Ajouté) ou `-` (Retiré).

---

4. Points Clés à Retenir

1. Utilisation de Bootstrap

- **Design propre et responsive** grâce aux classes Bootstrap :
  - `table-responsive` : Gère le défilement horizontal.
  - `table-bordered` et `table-striped` : Améliorent la lisibilité du tableau.
  - `table-dark` : Met en évidence l'en-tête.

2. Fonctionnement du `map`

- La méthode `.map` est utilisée pour transformer le tableau `transactions` en une série de lignes `<tr>`.
- Exemple de logique :
  - **Entrée** : `{ nom: "Salaire", montant: 3000, type: "+" }`.
  - **Sortie** : Une ligne `<tr>` avec :

```
<tr>
  <td>Salaire</td>
  <td>3000 MAD</td>
  <td>Ajouté</td>
</tr>
```

3. Calcul du Montant Absolu

- La méthode `Math.abs()` est utilisée pour afficher uniquement la valeur positive d'un montant, peu importe son signe.

4. Gestion Dynamique du Type

- Le type de transaction (`+` ou `-`) est traduit en texte compréhensible par l'utilisateur :
  - `Ajouté` pour `+`.
  - `Retiré` pour `-`.

Avantages et Optimisations

Avantages

1. **Réutilisable** : Le composant `Liste` peut être utilisé avec n'importe quel tableau de transactions (tant qu'il respecte la structure).
2. **Facile à lire** : Le tableau est clair et bien organisé grâce à Bootstrap.
3. **Responsive** : Le tableau s'adapte aux différents types d'écrans.

Optimisations Possibles

1. Ajouter une fonctionnalité pour trier ou filtrer les transactions selon le montant ou le type.
2. Ajouter un bouton pour supprimer une transaction spécifique.

App.js

Voici une explication complète et organisée du composant **App.js**, qui est le cœur de votre application de gestion d'argent de poche.

Fonctionnalité Principale

Le composant `App` :

1. Implémente le routage pour naviguer entre deux pages principales :
   - **Accueil** : Permet d'ajouter des transactions et d'afficher le solde total.
   - **Liste** : Affiche un tableau contenant toutes les transactions enregistrées.
2. Gère les données principales (états partagés) :
   - `solde` : Solde total d'argent disponible.
   - `transactions` : Liste des transactions enregistrées.

Structure et Fonctionnement

1. Gestion des États

- `solde` :
  Stocke le solde total.
  Initialisation :

  ```
  const [solde, setSolde] = useState(0);
  ```

  - Mise à jour lorsque l'utilisateur ajoute ou retire de l'argent.

- `transactions` :
  Stocke la liste des transactions (chaque transaction contient un nom, un montant, et un type).
  Initialisation :

```
const [transactions, setTransactions] = useState([]);
```

- Mise à jour lorsqu'une nouvelle transaction est enregistrée via `saveTransaction`.

---

2. La Fonction `saveTransaction`

Rôle

Cette fonction est appelée depuis le composant **Accueil** pour :

1. Calculer le nouveau solde total.
2. Ajouter une nouvelle transaction à la liste.

Détails

```
const saveTransaction = (nom, montant, type) => {
  const montantFinal = type === "+" ? parseFloat(montant) : -parseFloat(montant);
  setSolde(solde + montantFinal); // Mise à jour du solde.

  const newTransaction = { nom, montant: montantFinal, type };
  setTransactions([...transactions, newTransaction]); // Ajout à la liste des
transactions.
};
```

- **Calcul du Montant Final** :
  - Si le type est `"+"`, le montant est ajouté au solde.
  - Si le type est `"-"`, le montant est soustrait.
- **Ajout d'une Nouvelle Transaction** :
  - Un objet contenant les détails de la transaction est ajouté au tableau `transactions`.

---

3. Routage

Le routage est configuré à l'aide de **React Router** pour naviguer entre les composants `Accueil` et `Liste`.

Structure

```
<Router>
  <nav>...</nav> <!-- Barre de navigation -->
  <Routes>
    <Route path="/" element={<Accueil ... />} />
    <Route path="/liste" element={<Liste ... />} />
  </Routes>
</Router>
```

- **Composant** `Router` : Définit l'environnement de routage.
- **Composant** `Routes` : Regroupe toutes les routes de l'application.
- **Composant** `Route` :
  - `path` : Spécifie l'URL.
  - `element` : Indique le composant à afficher.

Liens de Navigation

La barre de navigation permet de passer d'une page à l'autre :

```
<nav className="navbar navbar-expand-lg navbar-dark bg-primary">
  <Link className="navbar-brand" to="/">Gestion d'Argent</Link>
  <ul>
    <li><Link className="nav-link" to="/">Accueil</Link></li>
    <li><Link className="nav-link" to="/liste">Liste</Link></li>
  </ul>
</nav>
```

- **Classe Bootstrap** : Ajoute des styles prédéfinis (navbar, couleurs, espacement).
- **Composant** `Link` : Crée des liens internes pour naviguer sans recharger la page.

---

4. Intégration des Composants

- **Accueil**

```
<Route
  path="/"
  element={<Accueil solde={solde} saveTransaction={saveTransaction} />}
/>
```

- La page d'accueil affiche :
  - Le solde total.
  - Un formulaire pour ajouter des transactions.
  - Appelle `saveTransaction` pour gérer les transactions.

- **Liste**

```
<Route
  path="/liste"
  element={<Liste transactions={transactions} />}
/>
```

- La page liste affiche un tableau contenant toutes les transactions enregistrées.

---

5. Design avec Bootstrap

Bootstrap est utilisé pour styliser les éléments :

- **Barre de navigation** :

```
<nav className="navbar navbar-expand-lg navbar-dark bg-primary">
```

  - `bg-primary` : Couleur bleu foncé pour la barre.
  - `navbar-dark` : Texte en couleur claire.
- **Espacement et conteneur principal** :

```
<div className="container mt-4">
```

  - `container` : Centrage et espacement automatique.
  - `mt-4` : Marges en haut.

Résumé des Flux

1. Ajout d'une Transaction (Accueil) :

1. L'utilisateur saisit les informations dans le formulaire.
2. `saveTransaction` est appelée avec les données.
3. Le solde et la liste des transactions sont mis à jour.

2. Affichage des Transactions (Liste) :

1. Le composant **Liste** reçoit la prop `transactions`.
2. Les transactions sont affichées dans un tableau avec Bootstrap.

3. Navigation :

- La navigation entre les pages est fluide grâce à **React Router**.

Points Forts du Code

1. **Séparation des Composants** :
   - `Accueil` et `Liste` ont des responsabilités distinctes.
2. **Centralisation des Données** :
   - Les états `solde` et `transactions` sont gérés dans `App` et partagés via des props.
3. **Design Moderne** :
   - Bootstrap est utilisé pour un style propre et responsive.

Améliorations Possibles

1. **Validation des Données** :
    - Vérifier que le montant est un nombre positif.
2. **Suppression des Transactions** :
    - Ajouter une fonctionnalité pour supprimer une transaction.
3. **Stockage Local** :
    - Utiliser **localStorage** pour sauvegarder les transactions entre les sessions.
