Étape 5 : Profil utilisateur avec useReducer

Objectif

Créer une page de profil avec un formulaire d'édition complexe géré par useReducer.

Concepts pratiqués

- useReducer pour les états complexes
- Pattern action / dispatch / reducer
- Formulaires avec validation
- Gestion des états multiples

Pourquoi useReducer plutôt que useState ?

useState	useReducer
État simple (booléen, string)	État complexe (objet, multiples champs)
Peu de mises à jour	Plusieurs types de mises à jour
Logique simple	Logique complexe, interdépendante

Exemple concret

```
// Avec useState : 5 setters, logique dispersée
const [name, setName] = useState('');
const [email, setEmail] = useState('');
const [phone, setPhone] = useState('');
const [isEditing, setIsEditing] = useState(false);
const [hasChanges, setHasChanges] = useState(false);

// Avec useReducer : logique centralisée
const [state, dispatch] = useReducer(reducer, initialState);
dispatch({ type: 'UPDATE_FIELD', field: 'name', value: 'Victor' });
```

À créer

1. Composant `ProfilePage.jsx`

La page de profil qui :

- Affiche les informations de l'utilisateur
- Permet de passer en mode édition
- Utilise useReducer pour gérer l'état

2. Composant `ProfileForm.jsx`

Le formulaire d'édition qui :

- Reçoit les données actuelles
- Permet de modifier tous les champs
- Valide les entrées
- Annule ou sauvegarde les modifications

Actions à implémenter

```
const actions = {
  SET_USER: 'SET_USER',           // Initialiser les données
  START_EDIT: 'START_EDIT',       // Passer en mode édition
  CANCEL_EDIT: 'CANCEL_EDIT',     // Annuler les modifications
  UPDATE_FIELD: 'UPDATE_FIELD',   // Modifier un champ
  SAVE: 'SAVE',                   // Sauvegarder
  SET_ERROR: 'SET_ERROR',         // Définir une erreur
};
```

Indices

▶ 💡 Structure du state

```
const initialState = {
  user: null,           // Données utilisateur
  editedUser: null,     // Données en cours d'édition
  isEditing: false,     // Mode édition actif ?
  isSaving: false,      // Sauvegarde en cours ?
  errors: {},           // Erreurs de validation
};
```

▶ 💡 Structure du reducer

```
function profileReducer(state, action) {
  switch (action.type) {
    case 'SET_USER':
      return {
        ...state,
        user: action.payload,
        editedUser: action.payload,
      };

    case 'START_EDIT':
      return {
```

```
        ...state,
        isEditing: true,
        editedUser: { ...state.user },  // Copie pour édition
        errors: {},
      };

    case 'UPDATE_FIELD':
      return {
        ...state,
        editedUser: {
          ...state.editedUser,
          [action.field]: action.value,
        },
      };

    // ... autres cases

    default:
      return state;
  }
}
```

Points d'attention

⚠ Immutabilité dans le reducer

```
// ❌ ERREUR : Mutation directe
case 'UPDATE_FIELD':
  state.editedUser[action.field] = action.value;  // NON !
  return state;

// ☑ CORRECT : Nouvelle référence
case 'UPDATE_FIELD':
  return {
    ...state,
    editedUser: {
      ...state.editedUser,
      [action.field]: action.value,
    },
  };
```

⚠ Actions bien structurées

```
// Structure recommandée
dispatch({
  type: 'UPDATE_FIELD',  // Que faire ?
  field: 'email',        // Sur quoi ?
```

```
  value: 'new@email.com' // Nouvelle valeur
});
```

Critères de validation

- [ ] Le mode édition s'active/désactive
- [ ] Les modifications sont visibles en temps réel
- [ ] L'annulation restaure les données originales
- [ ] La validation fonctionne
- [ ] Le reducer est une fonction pure (pas de mutation)