

Application Cliente : ReactJS

Introduction à ReactJS

ReactJS est une bibliothèque JavaScript développée par Facebook pour la création d'interfaces utilisateur (UI). Elle est principalement utilisée pour construire des applications web monopage (SPA) où la mise à jour de l'interface se fait de manière dynamique sans avoir à recharger la page entière. React est très populaire grâce à sa flexibilité, sa performance et sa capacité à offrir une expérience utilisateur fluide.

Objectif de ReactJS

L'objectif principal de React est de simplifier la création d'interfaces utilisateurs en permettant aux développeurs de décomposer l'UI en composants réutilisables. React permet une mise à jour efficace du DOM (Document Object Model), rendant ainsi les applications plus rapides et réactives.

Technologies associées

ReactJS repose sur plusieurs technologies et outils complémentaires :

- **JSX (JavaScript XML)** : Un langage de balisage syntaxique qui permet de décrire la structure de l'UI à l'aide de JavaScript.
- **Virtual DOM** : Un concept clé de React qui optimise la mise à jour de l'interface en ne modifiant que les parties du DOM qui changent.
- **Redux** : Une bibliothèque de gestion d'état souvent utilisée avec React pour gérer les états complexes des applications.
- **React Router** : Permet de gérer la navigation dans une application React monopage.

Utilisation de ReactJS

ReactJS est principalement utilisé pour développer des interfaces utilisateur complexes dans des applications web et mobiles. Il peut être utilisé dans les cas suivants :

- Création d'applications web monopage (SPA) *Single Page Application*.
- Applications de gestion de contenu.
- Tableaux de bord interactifs.
- Interfaces de gestion d'applications en temps réel (comme les messageries instantanées).
- Développement d'applications mobiles avec **React Native**.

Avantages de ReactJS

1. **Performance élevée** : Grâce au Virtual DOM, React rend les mises à jour plus efficaces en réduisant le nombre de manipulations directes du DOM.
2. **Composants réutilisables** : React permet de créer des composants UI réutilisables, ce qui améliore la modularité et la maintenance du code.
3. **Grande communauté** : Avec une grande base d'utilisateurs, React bénéficie de nombreuses bibliothèques, outils et ressources pour faciliter le développement.

4. **Compatibilité avec d'autres frameworks** : React peut être intégré à d'autres frameworks ou bibliothèques comme Redux pour la gestion de l'état.
5. **Développement multiplateforme** : Avec **React Native**, vous pouvez créer des applications mobiles pour iOS et Android en utilisant la même base de code JavaScript.

Inconvénients de ReactJS

1. **Courbe d'apprentissage** : Bien que React soit plus simple que certains autres frameworks, il peut avoir une courbe d'apprentissage, notamment en ce qui concerne JSX, les hooks et la gestion de l'état avec Redux.
2. **Besoin d'outils complémentaires** : Pour une utilisation complète, React nécessite souvent l'intégration d'autres bibliothèques comme Redux, React Router, ce qui peut augmenter la complexité du projet.
3. **Mise à jour du projet** : La communauté React évolue rapidement, ce qui peut entraîner des changements dans les meilleures pratiques ou des incompatibilités de version.
4. **SEO** : Les applications React ne sont pas indexées facilement par les moteurs de recherche par défaut. Cependant, des solutions comme le rendu côté serveur (SSR) peuvent résoudre ce problème.

Pourquoi choisir ReactJS ?

Vous devriez choisir ReactJS pour les projets qui nécessitent une interface utilisateur dynamique et réactive. Il est particulièrement adapté pour les applications où les performances et la modularité du code sont essentielles. De plus, si vous avez besoin de créer des applications mobiles en parallèle, React Native permet de réutiliser une grande partie du code.

Pour quel type de projet ReactJS est-il recommandé ?

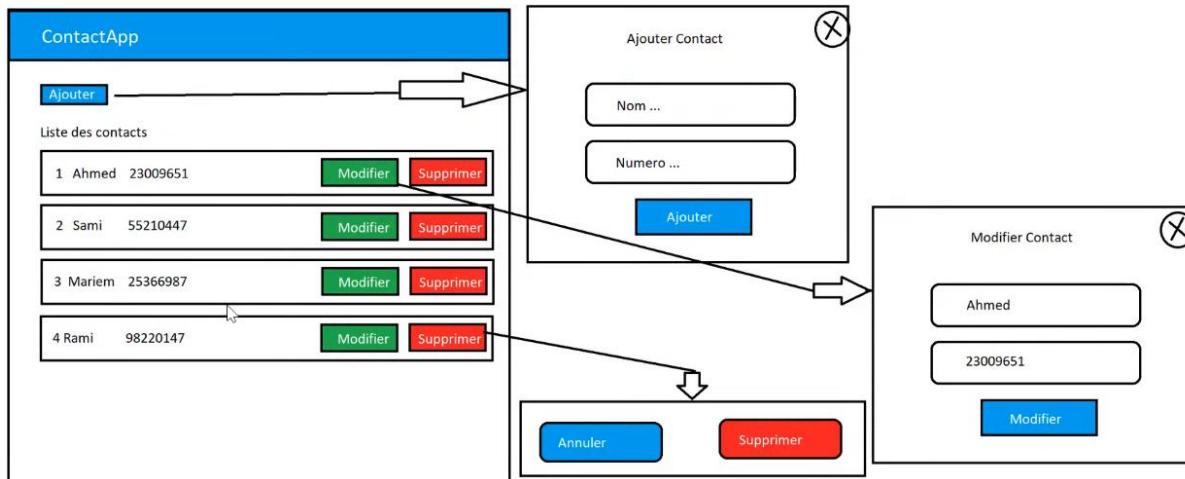
- **Applications monopage (SPA)** : Idéales pour des interfaces utilisateur dynamiques et fluides.
- **Applications interactives et temps réel** : Comme les messageries, les plateformes de collaboration ou les systèmes de notifications en temps réel.
- **Projets nécessitant une gestion d'état complexe** : Utilisation de Redux ou du Context API de React pour gérer des états globaux dans des applications complexes.
- **Applications mobiles (via React Native)** : Créer des applications mobiles pour Android et iOS avec une base de code partagée.

Tableau récapitulatif

Critère	ReactJS
Objectif	Création d'interfaces utilisateur dynamiques
Technologies associées	JSX, Virtual DOM, Redux, React Router, React Native
Avantages	Performance élevée, Composants réutilisables, Grande communauté, Développement multiplateforme
Inconvénients	Courbe d'apprentissage, Besoin d'outils complémentaires, Mise à jour rapide, SEO difficile
Recommandé	Applications web SPA, Applications mobiles (React Native), Projets

Critère	ReactJS
pour	interactifs en temps réel, Gestion d'états complexes
Choisir ReactJS	Si vous souhaitez créer des applications dynamiques, performantes, et évolutives avec une interface utilisateur réactive

ReactJS est donc un excellent choix pour les développeurs souhaitant créer des interfaces utilisateurs modernes, performantes et évolutives, notamment pour les applications web et mobiles



1. Créer un dossier « frontend » qui va recevoir l'application client qui va consommer les APIs développés coté serveur (Backend).

Lancer la commande suivante : `npx create-react-app frontend`

Le nom de l'application doit être minuscule

```
C:\Users\kamel\Desktop\Application>npx create-react-app frontend
Creating a new React app in C:\Users\kamel\Desktop\Application\frontend.
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

added 1458 packages in 2m
241 packages are looking for funding
```

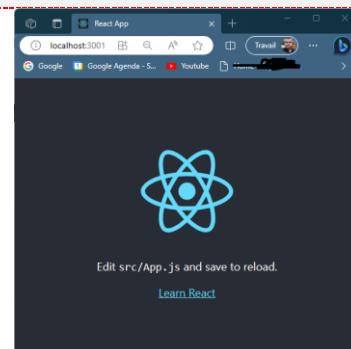
Figure 1: `npx create-react-app frontend`

2. Lancer les deux commandes suivantes :

```
cd frontend
npm start
```

```
C:\Users\kamel\Desktop\Application\frontend>npm start
> frontend@0.1.0 start
> react-scripts start
? Something is already running on port 3000.

Would you like to run the app on another port instead? » (Y/n)
```



L'ambition de React est de créer des interfaces utilisateurs, avec un outil rapide et modulaire. L'idée principale derrière React est que vous construisiez votre application à partir de composants. Un composant regroupe à la fois le HTML, le JS et le CSS, créés sur mesure pour vos besoins, et que vous pouvez réutiliser pour construire des interfaces utilisateurs.

Exercice 1: Composant Simple

Nom du fichier: App.js

```
import React from 'react';

function App() {
  return <h1>Bonjour, React!</h1>;
}
export default App;
```

Explication: Ce premier exercice est conçu pour introduire la création d'un composant simple dans React. Ici, App est un composant fonctionnel qui retourne un simple titre. Les composants fonctionnels sont une des façons les plus simples de créer des composants dans React, en utilisant une fonction JavaScript qui renvoie du JSX. Le JSX est une syntaxe proche de HTML mais utilisée dans le code JavaScript.

Exercice 2 : Utilisation des variables et types

1. **Objectif :** Afficher le nom d'un utilisateur et son âge.
2. **Instructions :**
 - Créez un composant UserInfo.
 - Dans ce composant, déclarez deux variables : name et age.
 - Affichez le message suivant dans le composant : "Nom : [name], Âge : [age]".
3. **Correction :**

```
import React from 'react';

function UserInfo() {
  const name = "Alice";
  const age = 30;

  return (
    <div>
      <p>Nom : {name}</p>
      <p>Âge : {age}</p>
    </div>
  );
}

export default UserInfo;
```

4. Appel de ce composant

Ajouter ce code dans index.js

```
import UserInfo from './UserInfo';
.....<UserInfo/>
```

Exercice 2 : Composants et props

1. **Objectif :** Créer un composant Greeting qui prend une prop name et affiche un message de bienvenue.
2. **Instructions :**
 - o Créez un composant Greeting.
 - o Ce composant prend une prop name et affiche : "Bienvenue, [name]!".
 - o Testez le composant dans un composant principal (App.js) en lui passant différentes valeurs pour name.
3. **Correction :**

```
import React from 'react';

function Greeting({ name }) {
  return <h1>Bienvenue, {name}!</h1>;
}

function App() {
  return (
    <div>
      <Greeting name="Alice" />
      <Greeting name="Bob" />
    </div>
  );
}

export default App;
```

Exercice 3 : Conditions

1. **Objectif :** Créer un composant UserStatus qui affiche un message selon l'âge de l'utilisateur.
2. **Instructions :**
 - o Créez un composant UserStatus.
 - o Dans ce composant, créez une variable age.
 - o Si age est inférieur à 18, affichez "Mineur". Sinon, affichez "Majeur".
3. **Correction :**

```
import React from 'react';

function UserStatus() {
  const age = 20;

  return (
    <div>
      {age < 18 ? <p>Mineur</p> : <p>Majeur</p>}
    </div>
  );
}

export default UserStatus;
```

Exercice 4 : Boucles avec map()

1. **Objectif :** Afficher une liste de noms.
2. **Instructions :**
 - o Créez un composant NameList.
 - o Dans ce composant, créez une liste de noms.
 - o Utilisez la méthode map() pour afficher chaque nom dans une balise dans une liste .
3. **Correction :**

```
import React from 'react';

function NameList() {
  const names = ["Alice", "Bob", "Charlie"];

  return (
    <ul>
      {names.map((name, index) => (
        <li key={index}>{name}</li>
      )))
    </ul>
  );
}

export default NameList;
```

Exercice 5 : Composants avec boucle et conditions

1. **Objectif :** Afficher une liste de produits avec disponibilité.
2. **Instructions :**
 - Créez un composant `ProductList`.
 - Dans ce composant, créez une liste de produits, chaque produit ayant un `name` et une propriété `inStock` (booléenne).
 - Affichez chaque produit dans une ``. Indiquez "En stock" ou "Rupture de stock" en fonction de la disponibilité.
3. **Correction :**

```
import React from 'react';

function ProductList() {
  const products = [
    { name: "Produit A", inStock: true },
    { name: "Produit B", inStock: false },
    { name: "Produit C", inStock: true },
  ];

  return (
    <ul>
      {products.map((product, index) => (
        <li key={index}>
          {product.name} - {product.inStock ? "En stock" : "Rupture de stock"}
        </li>
      )))
    </ul>
  );
}

export default ProductList;
```

Exercice 6 : Utilisation des props, conditions, et composants imbriqués

1. **Objectif :** Créer une liste d'utilisateurs avec des rôles affichés différemment.
2. **Instructions :**
 - Créez un composant `UserRole` qui prend une prop `role`.
 - Si le rôle est "admin", affichez "Administrateur". Sinon, affichez "Utilisateur".
 - Créez un composant `UserList` avec une liste d'objets utilisateurs (chaque utilisateur a un nom et un rôle).
 - Utilisez `UserRole` dans `UserList` pour chaque utilisateur.
3. **Correction :**

```
import React from 'react';

function UserRole({ role }) {
```

```

        return <span>{role === "admin" ? "Administrateur" : "Utilisateur"}</span>;
    }

function UserList() {
    const users = [
        { name: "Alice", role: "admin" },
        { name: "Bob", role: "user" },
        { name: "Charlie", role: "user" },
    ];

    return (
        <ul>
            {users.map((user, index) => (
                <li key={index}>
                    {user.name} - <UserRole role={user.role} />
                </li>
            ))}
        </ul>
    );
}

export default UserList;

```

Exercice 7: Affichage d'une Liste

Nom du fichier: App.js

```

import React from 'react';

function App() {
    const items = ['Pommes', 'Bananes', 'Cerises'];

    return (
        <ul>
            {items.map((item, index) => (
                <li key={index}>{item}</li>
            ))}
        </ul>
    );
}

export default App;

```

Explication: Cet exercice introduit les listes en React et montre comment utiliser la méthode `.map()` pour générer une série d'éléments JSX. Ici, chaque élément de la liste est rendu sous forme d'élément ``. La clé (`key`) unique pour chaque élément est importante dans React pour optimiser le rendu.

Exercice 8: Gestion de l'État avec useState

Nom du fichier: App.js

```

import React, { useState } from 'react';

function App() {
    const [count, setCount] = useState(0);

    const increment = () => setCount(count + 1);

    return (
        <div>
            <p>Compteur: {count}</p>
            <button onClick={increment}>Incrémenter</button>
        </div>
    );
}

```

```
export default App;
```

Explication: Cet exercice utilise le hook `useState`, qui permet aux composants fonctionnels de gérer l'état local. Ici, `count` est la valeur d'état et `setCount` est la fonction pour mettre à jour cette valeur. Chaque clic sur le bouton augmente le compteur de 1.

Exercice 9. Utiliser `useEffect` au Montage du Composant

Le code dans `useEffect` s'exécute **une seule fois**, lorsque le composant est monté.

N'oublier pas de changer le code de `index.js` comme suit:

```
root.render(
  <>
    <App />
    <UserInfo/>
    <UserStatus/>
  </>
);

);
```

Code de `App.js`:

```
import React, { useEffect } from 'react';

function App() {
  useEffect(() => {
    console.log('➊ Le composant est monté.');
  }, []); // Dépendances vides -> S'exécute uniquement au montage

  return <h1>Composant monté</h1>;
}

export default App;
```

Résumé :

- Les crochets de dépendances vides `[]` signifient que l'effet s'exécute uniquement lors du premier rendu.

Exercice 10 :Utiliser `useEffect` pour le Nettoyage (Cleanup)

Un nettoyage est effectué lors du démontage du composant.

N'oublier pas de changer le code de `index.js` comme suit:

```
<React.StrictMode>
  <App />
  <UserInfo/>
  <UserStatus/>
</React.StrictMode>
```

Code de `App.js` :

```
import React, { useEffect } from 'react';

function App() {
  useEffect(() => {
    console.log('➊ Le composant est monté.');
  }, []);
```

```

    return () => {
      console.log('🔴 Le composant est démonté.');
    }; // Retourne une fonction de nettoyage
  , []);
}

return <h1>Nettoyage avec useEffect</h1>;
}

export default App;

```

Résumé :

- La fonction retournée dans `useEffect` est exécutée **lors du démontage** du composant.

Exercice 11: Utiliser `useEffect` pour Réagir à un Changement

Le code s'exécute chaque fois qu'une variable spécifiée dans les dépendances change.

Exemple :

```

import React, { useState, useEffect } from 'react';

function App() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log(`🟡 Le compteur a changé : ${count}`);
  }, [count]); // Dépend de `count`

  return (
    <div>
      <p>Compteur : {count}</p>
      <button onClick={() => setCount(count + 1)}>Incrémenter</button>
    </div>
  );
}

export default App;

```

Résumé :

- `useEffect` réagit uniquement lorsque `count` change grâce à `[count]` dans les dépendances.

Exercice 12. Utiliser `useEffect` avec un Intervalle

Exécution répétée à intervalle régulier avec `setInterval` et nettoyage.

Exemple :

```

import React, { useState, useEffect } from 'react';

function App() {
  const [time, setTime] = useState(new Date().toLocaleTimeString());

  useEffect(() => {
    const intervalId = setInterval(() => {
      setTime(new Date().toLocaleTimeString());
    }, 1000);
  });

  return () => clearInterval(intervalId); // Nettoyage : arrêter l'intervalle
}

```

```

    }, []);
    return <p>Heure actuelle : {time}</p>;
}

export default App;

```

Résumé :

- Le `setInterval` met à jour l'état toutes les secondes.
- Le `clearInterval` dans le nettoyage empêche les fuites de mémoire.

Exercice 13. Utiliser `useEffect` pour Charger des Données

Simuler une requête réseau lors du montage.

Exemple :

```

import React, { useState, useEffect } from 'react';

function App() {
  const [data, setData] = useState(null);

  useEffect(() => {
    console.log('Chargement des données...');

    setTimeout(() => {
      setData('Données chargées !');
      console.log('✅ Données chargées.');
    }, 2000); // Simule un délai de 2 secondes
  }, []);

  return <p>{data || 'Chargement en cours...'}</p>;
}

export default App;

```

Résumé :

- `useEffect` est idéal pour les appels réseau ou les tâches asynchrones.
- Les crochets vides `[]` garantissent que cela ne s'exécute qu'une fois.

14. Exercice : Affichage d'un message à chaque rendu

Objectif : Apprendre à utiliser `useEffect` pour afficher un message dans la console à chaque rendu.

- Créez un composant React appelé `MessageLogger`.
- Utilisez `useEffect` pour afficher "Le composant a été rendu" dans la console à chaque fois que le composant est rendu.
- Observez le comportement dans la console lorsque le composant est monté, mis à jour et démonté.

Correction :

```

import React, { useEffect } from 'react';

function MessageLogger() {
  useEffect(() => {
    console.log('Le composant a été rendu');
  });
}

```

```

    return <div>Vérifiez la console pour voir le message de rendu</div>;
}

export default MessageLogger;

```

15. Exercice : Exécution d'un effet uniquement lors du montage (exercice optionnel)

Objectif : Utiliser `useEffect` pour exécuter un effet uniquement lorsque le composant est monté.

- Créez un composant `WelcomeMessage`.
- Utilisez `useEffect` pour afficher "Bienvenue!" dans la console uniquement lorsque le composant est monté (c'est-à-dire, au premier rendu).
- Utilisez le tableau de dépendances vide (`[]`) pour contrôler cet effet.

Correction :

```

import React, { useEffect } from 'react';

function WelcomeMessage() {
  useEffect(() => {
    console.log('Bienvenue!');
  }, []); // Exécution unique lors du montage

  return <div>Message de bienvenue affiché dans la console</div>;
}

export default WelcomeMessage;

```

16. Exercice : Compteur avec mise à jour de titre

Objectif : Utiliser `useEffect` pour mettre à jour le titre de la page à chaque modification du compteur.

- Créez un composant `Counter`.
- Ajoutez un état `count` avec `useState` pour gérer un compteur.
- Utilisez `useEffect` pour mettre à jour le titre de la page avec la valeur du compteur chaque fois que le compteur change.
- Ajoutez un bouton pour incrémenter le compteur.

Correction :

```

import React, { useState, useEffect } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `Compteur : ${count}`;
  }, [count]); // Mise à jour uniquement quand count change

  return (
    <div>
      <p>Compteur : {count}</p>
      <button onClick={() => setCount(count + 1)}>Incrémenter</button>
    </div>
  );
}

```

```
export default Counter;
```

17. Exercice : Récupération de données avec `fetch`

Objectif : Utiliser `useEffect` pour effectuer une requête HTTP et afficher les données.

- Créez un composant `UserList`.
- Utilisez `useEffect` pour récupérer des données depuis l'API suivante : <https://jsonplaceholder.typicode.com/users>.
- Stockez les données dans un état `users` avec `useState` et affichez la liste des utilisateurs.
- Assurez-vous que la requête ne soit exécutée qu'une seule fois au montage du composant.

Correction :

```
import React, { useState, useEffect } from 'react';

function UserList() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(response => response.json())
      .then(data => setUsers(data))
      .catch(error => console.error('Erreur:', error));
  }, []); // Exécution unique lors du montage

  return (
    <div>
      <h3>Liste des utilisateurs</h3>
      <ul>
        {users.map(user => (
          <li key={user.id}>{user.name}</li>
        ))}
      </ul>
    </div>
  );
}

export default UserList;
```

18. Exercice : Déclenchement d'un effet en fonction d'une variable

Objectif : Utiliser `useEffect` pour déclencher un effet uniquement lorsque la valeur d'une variable spécifique change.

- Créez un composant `SearchUser`.
- Ajoutez un champ de saisie pour rechercher un utilisateur et stockez la valeur dans un état `query`.
- Utilisez `useEffect` pour afficher "Recherche en cours..." dans la console chaque fois que la valeur de `query` change.

Correction :

```
import React, { useState, useEffect } from 'react';

function SearchUser() {
  const [query, setQuery] = useState('');
```

```

useEffect(() => {
  if (query) {
    console.log('Recherche en cours...');

  }
}, [query]); // Exécution quand query change

return (
  <div>
    <input
      type="text"
      value={query}
      onChange={(e) => setQuery(e.target.value)}
      placeholder="Rechercher un utilisateur"
    />
  </div>
);
}

export default SearchUser;

```

Exercice 19: Gestion des événements

Nom du fichier: handleChange.js

```

import React, { useState } from 'react';

function HandleChange() {
  const [input, setInput] = useState('');

  const handleChange = (event) => {
    setInput(event.target.value);
  };

  return (
    <div>
      <input type="text" value={input} onChange={handleChange} />
      <p>Vous avez écrit: {input}</p>
    </div>
  );
}

export default HandleChange;

```

Explication: Cet exercice introduit la gestion des événements en React. Ici, un champ de saisie est lié à l'état local `input`. Chaque changement dans le champ appelle `handleChange`, qui met à jour `input` avec la nouvelle valeur. La saisie est affichée en temps réel sous le champ.

Exercice 20: Formulaire avec Validation

Nom du fichier: ValidatedForm.js

```

import React, { useState } from 'react';

function ValidatedForm() {
  const [input, setInput] = useState('');
  const [error, setError] = useState('');

  const handleChange = (event) => {
    setInput(event.target.value);
    setError('');
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    if (input === '') {

```

```

        setError('Ce champ est obligatoire');
    } else {
        alert('Formulaire soumis');
    }
};

return (
    <form onSubmit={handleSubmit}>
        <input
            type="text"
            value={input}
            onChange={handleChange}
        />
        {error && <p style={{ color: 'red' }}>{error}</p>}
        <button type="submit">Soumettre</button>
    </form>
);
}

export default ValidatedForm;

```

Explication: Ce code montre comment valider un formulaire avec une vérification basique en React. `handleSubmit` empêche le comportement par défaut de soumission du formulaire avec `event.preventDefault()`. Si le champ est vide, un message d'erreur s'affiche. Sinon, une alerte signale la soumission.

Exercice 21: Envoi d'une requête HTTP avec `fetch`

Nom du fichier: FetchData.js

```

import React, { useState, useEffect } from 'react';

function FetchData() {
    const [data, setData] = useState(null);

    useEffect(() => {
        fetch('https://jsonplaceholder.typicode.com/users')
            .then((response) => response.json())
            .then((data) => setData(data));
    }, []);

    return (
        <div>
            {data ? <pre>{JSON.stringify(data, null, 2)}</pre> : <p>Chargement...</p>}
        </div>
    );
}

export default FetchData;

```

Explication: Cet exercice montre comment récupérer des données depuis une API externe avec `fetch`. Dans `useEffect`, `fetch` est appelé une fois, puis les données sont stockées dans `data`. Si `data` n'est pas encore disponible, un message de chargement s'affiche.

Exercice 22: Réutilisation des composants

Nom du fichier: Alert.js et App.js

Fichier Alert.js:

```

import React from 'react';

function Alert({ message, type }) {
    const styles = {

```

```

    success: { color: 'green' },
    error: { color: 'red' },
    info: { color: 'blue' },
};

return <p style={styles[type]}>{message}</p>;
}

export default Alert;

```

Fichier App.js:

```

import React from 'react';
import Alert from './Alert';

function App() {
  return (
    <div>
      <Alert message="Succès!" type="success" />
      <Alert message="Erreur!" type="error" />
      <Alert message="Information!" type="info" />
    </div>
  );
}

export default App;

```

Explication: Ici, nous créons un composant réutilisable Alert qui affiche différents messages selon le type. Chaque type (success, error, info) reçoit une couleur distincte pour se démarquer visuellement. Le composant est ensuite utilisé dans App.

Exercice 20: React Router (Navigation entre les pages)

npm install react-router-dom

Structure du projet

```

src/
  └── App.js
  └── Home.js
  └── About.js
  └── index.js

```

Home.js

Un composant simple pour la page d'accueil :

```

import React from "react";

function Home() {
  return <h1>Bienvenue sur la page d'accueil</h1>;
}

export default Home;

```

About.js

Un composant simple pour la page "À propos" :

```
import React from "react";

function About() {
    return <h1>À propos de notre site</h1>;
}

export default About;
```

Nom du fichier: App.js

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';

function Home() {
    return <h2>Accueil</h2>;
}

function About() {
    return <h2>À propos</h2>;
}

function App() {
    return (
        <Router>
            <div>
                <nav>
                    <Link to="/">Accueil</Link> |
                    <Link to="/about">À propos</Link>
                </nav>

                <Routes>
                    <Route path="/" element={<Home />} />
                    <Route path="/about" element={<About />} />
                </Routes>
            </div>
        </Router>
    );
}

export default App;
```

Explication: Cet exercice introduit React Router pour gérer la navigation entre plusieurs pages. Link est utilisé pour naviguer sans recharger la page, tandis que Route détermine quel composant afficher pour chaque chemin (/ pour Home et /about pour About).

Erreur : supprimer le dossier node_modules et relancer la commande npm install

Exercice 21 : Créer une page avec Header, Side, Main, et Footer

1. **Objectif :** Construire une application avec plusieurs composants. L'application doit afficher une page structurée en sections, avec un contenu dynamique dans Main.
2. **Instructions :**
 - Créez les composants suivants :
 - Header : pour le haut de la page avec un titre et une barre de navigation.
 - Side : pour un menu latéral avec des liens.
 - Main : pour le contenu principal de la page, qui change dynamiquement.

- Footer : pour le bas de la page avec des informations sur le site.
- Utilisez App.js pour faire appel à ces composants et afficher la page complète.
- Implémentez un système pour mettre à jour dynamiquement le contenu de Main lorsqu'un utilisateur clique sur un lien dans Side.

Étapes de réalisation

Étape 1 : Structure de base dans App.js

Créez un fichier App.js qui importera et organisera tous les composants.

```
// src/App.js
import React, { useState } from 'react';
import Header from './components/Header';
import Side from './components/Side';
import Main from './components/Main';
import Footer from './components/Footer';

function App() {
  const [content, setContent] = useState("Bienvenue sur notre site !");

  const handleContentChange = (newContent) => {
    setContent(newContent);
  };

  return (
    <div>
      <Header />
      <div style={{ display: 'flex' }}>
        <Side onContentChange={handleContentChange} />
        <Main content={content} />
      </div>
      <Footer />
    </div>
  );
}

export default App;
```

Étape 2 : Création des composants Header, Side, Main, et Footer

1. Header.js : Affiche le titre de la page.

```
// src/components/Header.js
import React from 'react';

function Header() {
  return (
    <header style={{ padding: '10px', backgroundColor: '#282c34', color: 'white', textAlign: 'center' }}>
      <h1>Mon Site React</h1>
    </header>
  );
}

export default Header;
```

2. Side.js : Affiche un menu latéral avec des liens pour changer le contenu principal.

```
// src/components/Side.js
import React from 'react';

function Side({ onContentChange }) {
  return (
    <aside style={{ width: '200px', padding: '10px', backgroundColor: '#f5f5f5' }}>
      <h3>Menu</h3>
```

```

<ul>
    <li onClick={() => onContentChange("Accueil")}>Accueil</li>
    <li onClick={() => onContentChange("À propos de nous")}>À propos</li>
    <li onClick={() => onContentChange("Services")}>Services</li>
    <li onClick={() => onContentChange("Contactez-nous")}>Contact</li>
</ul>
</aside>
);
}

export default Side;

```

3. Main.js : Affiche le contenu principal, qui est mis à jour dynamiquement en fonction de ce qui est sélectionné dans Side.

```

// src/components/Main.js
import React from 'react';

function Main({ content }) {
    return (
        <main style={{ flex: 1, padding: '20px' }}>
            <h2>Contenu principal</h2>
            <p>{content}</p>
        </main>
    );
}

export default Main;

```

4. Footer.js : Affiche le bas de la page avec une mention ou un copyright.

```

// src/components/Footer.js
import React from 'react';

function Footer() {
    return (
        <footer style={{ padding: '10px', backgroundColor: '#282c34', color: 'white', textAlign: 'center', marginTop: '20px' }}>
            <p>© 2024 Mon Site React. Tous droits réservés. </p>
        </footer>
    );
}

export default Footer;

```

Résultat attendu

- Page avec structure complète :** App.js assemble chaque composant pour former une page structurée avec Header, Side, Main, et Footer.
- Contenu dynamique :** Lorsque l'utilisateur clique sur un lien dans Side, le contenu de Main est mis à jour dynamiquement.
- Style basique :** Ajoutez du style simple pour mettre en forme les composants et voir clairement chaque section.

3. Ajouter le framework CSS Bootstrap

<https://react-bootstrap.github.io/docs/getting-started/introduction>

```
npm install react-bootstrap bootstrap
```

Voici un exemple simple d'utilisation de **Bootstrap** avec des éléments comme du texte, des boutons et des images dans un projet React. Ce code montre comment intégrer Bootstrap pour structurer une page avec ces éléments.

Étapes préliminaires

- Installer Bootstrap** : Si vous ne l'avez pas encore installé, vous pouvez ajouter Bootstrap à votre projet en exécutant la commande suivante dans votre terminal :

```
npm install bootstrap
```

- Ensuite, vous devez importer Bootstrap dans le fichier `index.js` ou `App.js` :

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

Exemple de code avec Bootstrap :

```
import React from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';

function BootstrapExample() {
  return (
    <div className="container mt-5">
      {/* Titre */}
      <h1 className="text-center text-primary mb-4">Bienvenue dans React avec Bootstrap</h1>

      {/* Texte */}
      <p className="lead text-muted">
        Bootstrap est un framework CSS populaire qui permet de créer rapidement des mises en page et des composants modernes et réactifs.
      </p>

      {/* Bouton */}
      <div className="text-center mb-4">
        <button className="btn btn-success btn-lg">Cliquez ici</button>
      </div>

      {/* Image */}
      <div className="text-center">
        
      </div>
    </div>
  );
}

export default BootstrapExample;
```

Explication du code :

- container mt-5** : Utilisé pour créer un espace autour du contenu avec la classe `container` de Bootstrap, et `mt-5` pour ajouter une marge au-dessus.
- text-center** : Cette classe est utilisée pour centrer le texte.
- text-primary** : Applique la couleur primaire définie par Bootstrap au texte.
- lead et text-muted** : `lead` est utilisé pour agrandir un peu le texte et `text-muted` pour assombrir légèrement le texte.
- btn btn-success btn-lg** : Applique les classes Bootstrap pour styliser un bouton avec la couleur `btn-success` (vert) et `btn-lg` pour un bouton plus grand.

6. **img-fluid** : Applique à l'image une largeur flexible pour qu'elle s'adapte à l'écran.
7. **rounded shadow-lg** : Applique un bord arrondi et une ombre autour de l'image pour un effet visuel agréable.

Essayer d'appliquer ce thème dans l'application client (Frontend)

The screenshot shows a React application interface. At the top, a blue header bar displays "Contacts Manager V2.0". Below it, the main content area has two sections: "Liste des contacts" on the left and "Ajouter un contact" on the right. The "Liste des contacts" section contains a table with two rows. The first row has columns: # (1), Nom (Mark), Phone (66585587), Email (mark@example.com), and Actions (Modifier, Supprimer). The second row has similar columns. The "Ajouter un contact" section has input fields for Nom, Phone, and Email, followed by a "Ajouter" button.

#	Nom	Phone	Email	Actions
1	Mark	66585587	mark@example.com	<button>Modifier</button> <button>Supprimer</button>
2	Jacob	867857456	jacob@example.com	<button>Modifier</button> <button>Supprimer</button>

4. Vérifier Le code source de fichier app.js

Étape 1 : Création de la Structure de l'Application

Nous allons commencer par créer le composant principal `App` et ajouter un en-tête pour la navigation.

1. Configurez `App.js` :

- o Ouvrez `src/App.js`.
- o Importez `Container`, `Navbar`, et `Nav` de `react-bootstrap`.
- o Ajoutez une barre de navigation en utilisant le composant `Navbar` pour afficher le titre de l'application.

Code de base dans `App.js` :

```
import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import { Navbar, Nav, Container } from 'react-bootstrap';

function App() {
  return (
    <div className="App">
      <Navbar bg="primary" variant="dark">
        <Container>
          <Navbar.Brand href="#">Contacts Manager</Navbar.Brand>
          <Nav className="me-auto">
            <Nav.Link href="#">V2.0</Nav.Link>
          </Nav>
        </Container>
      </Navbar>
    </div>
  );
}

export default App;
```

- o **Explication** : Le `Navbar` utilise `bg="primary"` pour une couleur de fond bleue et `variant="dark"` pour un texte blanc. Nous avons un `Navbar.Brand` pour le titre et un lien de version.

Étape 2 : Afficher la Liste des Contacts

1. Ajouter un tableau pour les contacts :

- Ajoutez des composants Table et Row de react-bootstrap.
- Créez un tableau basique qui affichera les informations de chaque contact : Nom, Téléphone, Email, et Actions.

2. Code pour afficher une liste simple de contacts :

Dans App.js, ajoutez le tableau suivant sous la barre de navigation :

```
import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import { Navbar, Nav, Container, Col, Row, Table } from 'react-bootstrap';

function App() {
  return (
    <div className="App">
      <Navbar bg="primary" variant="dark">
        <Container>
          <Navbar.Brand href="#home">Contacts Manager</Navbar.Brand>
          <Nav className="me-auto">
            <Nav.Link href="#home">V2.0</Nav.Link>
          </Nav>
        </Container>
      </Navbar>

      <Container>
        <h2 className="mt-4">Liste des contacts</h2>
        <Table striped bordered hover>
          <thead>
            <tr>
              <th>#</th>
              <th>Nom</th>
              <th>Phone</th>
              <th>Email</th>
              <th>Actions</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td>1</td>
              <td>Mark</td>
              <td>66585587</td>
              <td>mark@example.com</td>
              <td>
                <button className="btn btn-success me-2">Modifier</button>
                <button className="btn btn-danger">Supprimer</button>
              </td>
            </tr>
            <tr>
              <td>2</td>
              <td>Jacob</td>
              <td>867857456</td>
              <td>jacob@example.com</td>
              <td>
                <button className="btn btn-success me-2">Modifier</button>
                <button className="btn btn-danger">Supprimer</button>
              </td>
            </tr>
          </tbody>
        </Table>
      </Container>
    </div>
  );
}

export default App;
```

- **Explication :** Ce tableau est affiché à l'aide de Table de React-Bootstrap, et contient une liste d'exemples de contacts pour tester l'interface.

Étape 3 : Ajouter un Formulaire pour les Nouveaux Contacts

1. Ajoutez un formulaire pour créer un contact :

- Utilisez Form, FormGroup, Row, Col, et Button de react-bootstrap.
- Ajoutez les champs pour le nom, le téléphone et l'email.

2. Code pour le formulaire d'ajout de contact :

Ajoutez ce code sous le tableau :

```
import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import { Navbar, Nav, Container, Col, Row, Table } from 'react-bootstrap';

function App() {
  return (
    <div className="App">
      {/* Code de la barre de navigation */}
      <Container>
        <Row>
          <Col lg={8}>
            <h2>Liste des contacts</h2>
            {/* Code de la table */}
          </Col>
          <Col lg={4}>
            <h2 className="mt-4" >Ajouter un contact</h2>
            <Form>
              <FormGroup as={Row} controlId="nom">
                <Form.Label column sm={3}>Nom</Form.Label>
                <Col sm={9}>
                  <Form.Control type="text" placeholder="Entrez le nom" />
                </Col>
              </FormGroup>

              <FormGroup as={Row} controlId="phone">
                <Form.Label column sm={3}>Phone</Form.Label>
                <Col sm={9}>
                  <Form.Control type="tel" placeholder="Entrez le numéro" />
                </Col>
              </FormGroup>

              <FormGroup as={Row} controlId="email">
                <Form.Label column sm={3}>Email</Form.Label>
                <Col sm={9}>
                  <Form.Control type="email" placeholder="Entrez l'email" />
                </Col>
              </FormGroup>

              <Button variant="primary" type="submit" className="mt-3">
                Ajouter
              </Button>
            </Form>
          </Col>
        </Row>
      </Container>
    </div>
  );
}

export default App;
```

- **Explication** : Le formulaire est conçu pour accepter les informations de contact et est placé à droite du tableau. Le Button envoie les données, mais pour l'instant, il ne fait rien.

Étape 4 : Ajouter les Modals pour Modifier et Supprimer les Contacts

1. Ajoutez des boutons d'action avec des modals :

- Ajoutez des modals pour l'édition et la suppression.
- Utilisez useState pour gérer l'affichage et la fermeture des modals.

2. Code pour les modals d'édition et de suppression :

Ajoutez les imports et l'état dans App.js :

```
import { useState } from 'react';
import { Modal, Button } from 'react-bootstrap';

function App() {
  const [showEdit, setShowEdit] = useState(false);
  const [showDelete, setShowDelete] = useState(false);

  const handleEditClose = () => setShowEdit(false);
  const handleEditShow = () => setShowEdit(true);

  const handleDeleteClose = () => setShowDelete(false);
  const handleDeleteShow = () => setShowDelete(true);

  return (
    <div className="App">
      {/* Code de la barre de navigation et du contenu */}
      {/* Modal Modifier */}
      <Modal show={showEdit} onHide={handleEditClose}>
        <Modal.Header closeButton>
          <Modal.Title>Modifier contact</Modal.Title>
        </Modal.Header>
        <Modal.Body>
          {/* Formulaire de modification */}
        </Modal.Body>
        <Modal.Footer>
          <Button variant="secondary" onClick={handleEditClose}>Fermer</Button>
          <Button variant="primary" onClick={handleEditClose}>Sauvegarder</Button>
        </Modal.Footer>
      </Modal>

      {/* Modal Supprimer */}
      <Modal show={showDelete} onHide={handleDeleteClose}>
        <Modal.Header closeButton>
          <Modal.Title>Supprimer contact</Modal.Title>
        </Modal.Header>
        <Modal.Body>Voulez-vous vraiment supprimer ce contact ?</Modal.Body>
        <Modal.Footer>
          <Button variant="secondary" onClick={handleDeleteClose}>Fermer</Button>
          <Button variant="primary" onClick={handleDeleteClose}>Supprimer</Button>
        </Modal.Footer>
      </Modal>
    </div>
  );
}


```

Modals permettent à l'utilisateur de modifier ou supprimer un contact via un modal. Le gestionnaire d'état useState gère l'ouverture et la fermeture des modals.

Étape 5 : Ajouter l'action au bouton modifier et supprimer

```
.....
<button className="btn btn-success me-2" onClick={handleEditShow}>Modifier</button>
<button className="btn btn-danger" onClick={handleDeleteShow}>Supprimer</button>
.....
<button className="btn btn-success me-2" onClick={handleEditShow}>Modifier</button>
<button className="btn btn-danger" onClick={handleDeleteShow}>Supprimer</button>
.....
```



Redux

Votre application spécifie les actions associées à chaque composant, lesquelles définissent l'état du composant stocké dans un magasin (store) pour maintenir la vue à jour. Cependant, l'inconvénient réside dans le fait qu'un magasin est attribué à chaque composant, ce qui peut être limitant dans certaines applications, même si cela fonctionne bien pour React. Pour remédier à cela, Dan Abramov a introduit Redux en juin 2015, simplifiant la gestion du magasin en consolidant tous les états dans un seul magasin pour l'ensemble de l'application. Ainsi, tous les composants peuvent accéder aux données de manière plus globale.

Dans une application React utilisant Redux, les dossiers `reducers`, `actions`, et la configuration du `store` jouent des rôles spécifiques dans l'organisation de votre code.

Etape 6: Installer Redux avec la commande suivante :

```
npm i redux
```

5. Installer react-redux

```
npm i react-redux
```

6. installer redux-thunk

```
npm i redux-thunk
```

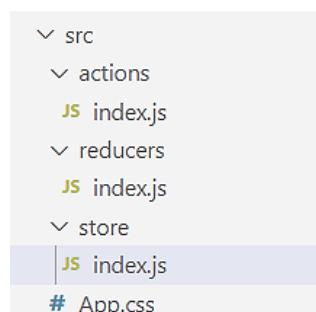
En résumé :

- redux : Gère l'état global de l'application.
- react-redux : Connecte Redux à vos composants React.
- redux-thunk : Ajoute la gestion des actions asynchrones dans Redux.

7. Ajouter les dossiers suivants dans le dossier « src »

- actions
- reducers
- store

Dans chaque ces trois dossiers, créer un fichier « index.js »



8. Installer axios (package pour consommer les APIs)

```
npm i axios
```

9. Ajouter les fichiers suivants :

src/actions/constantes.js (Dans ce fichier on enregistre les actions possibles que sera utilisées par les autres fichiers)

```
export const contactesConstants = {
  GET_ALL_CONTACTS_REQUEST: 'GET_ALL_CONTACTS_REQUEST',
  GET_ALL_CONTACTS_SUCCESS: 'GET_ALL_CONTACTS_SUCCESS',
```

```
    GET_ALL_CONTACTS_FAILURE: 'GET_ALL_CONTACTS_FAILURE'
}
```

src/actions/contact.actions.js

```
import axios from "axios";
import { contactesConstants } from "./constantes";

export const ListerContacts= ()=>{
    return async dispatch =>{
        dispatch({type:contactesConstants.GET_ALL_CONTACTS_REQUEST})

        try {
            const res = await axios.get('http://127.0.0.1:8000/contact/lister')
            if(res.status==200){
                dispatch({
                    type:contactesConstants.GET_ALL_CONTACTS_SUCCESS,
                    payload: {contacts: res.data})
                }
            } catch (error) {
                dispatch({
                    type:contactesConstants.GET_ALL_CONTACTS_FAILURE,
                    payload:{ error:error.res}})
            }
        }
    }
}
```

src/actions/index.js

```
export * from './contact.actions';
```

src/reducers/contact.reducer.js

```
// reducers/contact.reducer.js
import axios from "axios";
import { contactesConstants } from "../actions/constantes";

// État initial
const initialState = {
    contacts: [],
    loading: false,
    error: null,
};

// Le reducer pour les contacts
const contactReducer = (state = initialState, action) => {
    switch (action.type) {
        case contactesConstants.GET_ALL_CONTACTS_REQUEST:
            return { ...state, loading: true };
        case contactesConstants.GET_ALL_CONTACTS_SUCCESS:
            return { ...state, loading: false, contacts: action.payload.contacts };
        case contactesConstants.GET_ALL_CONTACTS_FAILURE:
            return { ...state, loading: false, error: action.payload.error };
        default:
            return state;
    }
};

export default contactReducer;

// L'action pour récupérer les contacts
export const ListerContacts = () => {
    return async (dispatch) => {
        dispatch({ type: contactesConstants.GET_ALL_CONTACTS_REQUEST });

        try {
```

```
const res = await axios.get('http://127.0.0.1:8000/contact/lister');
if (res.status === 200) {
    dispatch({
        type: contactesConstants.GET_ALL_CONTACTS_SUCCESS,
        payload: { contacts: res.data },
    });
}
} catch (error) {
    dispatch({
        type: contactesConstants.GET_ALL_CONTACTS_FAILURE,
        payload: { error: error.response },
    });
}
};
```

Voici le code reduces/ index.js

```
import contactReducer from './contact.reducer';
import { combineReducers } from 'redux';

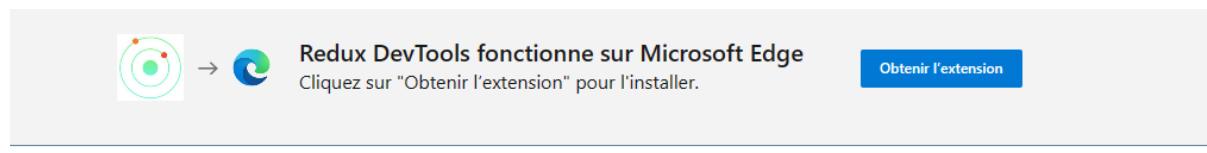
const rootReducer = combineReducers ({
    contact : contactReducer
})

export default rootReducer;
```

Voici le code store/index.js

```
import { applyMiddleware, legacy_createStore as createStore, legacy_createStore } from "redux"
import rootReducer from "../reducers"
import { thunk } from 'redux-thunk';
const store= legacy_createStore(
  rootReducer,
  applyMiddleware(thunk)
)
export default legacy_createStore;
```

Ajouter l'extension suivante :



10. Changer le code du fichier store/index.js comme suit :

```
import { createStore, applyMiddleware, compose } from 'redux'; // Import
createStore from 'redux';
import rootReducer from '../reducers';
import { thunk } from 'redux-thunk';
const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
```

```
const store = createStore(
  rootReducer,
  composeEnhancers(applyMiddleware(thunk))
);

export default store;
```

Source : <https://github.com/zalmoxisus/redux-devtools-extension>

11. Installer cors dans le backend (**serveur**) pour que les autres applications peuvent consommer les APIs

```
npm i cors
```

12. Modifier le code du fichier Serveur\app.js comme suit

```
const express = require('express')

const app = express()
const port = 8000
app.use(express.json())

const cors = require('cors');// ancienne version
import cors from 'cors';
app.use(cors());
```

13. Ajouter le code suivant dans le fichier Application\frontend\src\App.js

```
import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css'
import { useEffect } from 'react';
import { useDispatch } from 'react-redux';
import { ListerContacts } from './actions/contact.actions';

function App() {

  const dispatch = useDispatch();

  useEffect(()=>{
    dispatch(ListerContacts());
    console.log('Test');
  },[]);

  return (
    <div className="App">
      <Navbar bg="primary" data-bs-theme="dark">
        ...
    </div>
  );
}

export default App;
```

14. Modifier le fichier « index.js » de l'application frontend

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { Provider } from 'react-redux';
import store from './store';

const root = ReactDOM.createRoot(document.getElementById('root'));
```

```

root.render(
  <Provider store={store}>
    <React.StrictMode>
      <App />
    </React.StrictMode>
  </Provider>

);
reportWebVitals();

```

15. Changer le code du fichier src\App.js

```

import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import { Modal, Navbar, Table, Nav, Form, Row, Col, Button, Container } from
'react-bootstrap';
import { useState } from 'react';
import { useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { ListerContacts } from './actions/contact.actions';

function App() {
  const [showEdit, setShowEdit] = useState(false);
  const [showDelete, setShowDelete] = useState(false);
  const [currentContact, setCurrentContact] = useState(null);
  const [formData, setFormData] = useState({ name: '', phone: '', email: '' });

  const handleEditClose = () => setShowEdit(false);

  const handleEditShow = (contact) => {
    setCurrentContact(contact);
    setFormData({ name: contact.name, phone: contact.phone, email: contact.email });
    setShowEdit(true);
  };

  const handleDeleteClose = () => setShowDelete(false);
  const handleDeleteShow = (contact) => {
    setCurrentContact(contact);
    setShowDelete(true);
  };

  const handleEditChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleEditSave = () => {
    // Dispatch an action to update the contact (you need to implement the update
    // action)
    console.log('Saving edited contact', formData);
    setShowEdit(false);
  };

  const handleDeleteConfirm = () => {
    // Dispatch an action to delete the contact (you need to implement the delete
    // action)
    console.log('Deleting contact', currentContact);
    setShowDelete(false);
  };

  const dispatch = useDispatch();
  const contacts = useSelector(state => state.contact.contacts);

  useEffect(() => {
    dispatch(ListerContacts());
  }, [dispatch]);
}

return (
  <div className="App">
    <Navbar bg="primary" variant="dark">
      <Container>
        <Navbar.Brand href="#home">Contacts Manager</Navbar.Brand>

```

```

<Nav className="me-auto">
  <Nav.Link href="#home">V2.0</Nav.Link>
</Nav>
</Container>
</Navbar>
<Container>
  <Row>
    <Col lg={8}>
      <h2 className="mt-4">Liste des contacts</h2>
      {contacts ?
        <Table striped bordered hover>
          <thead>
            <tr>
              <th>#</th>
              <th>Nom</th>
              <th>Phone</th>
              <th>Email</th>
              <th>Actions</th>
            </tr>
          </thead>
          <tbody>
            {contacts.map((contact, index) =>
              <tr key={index}>
                <td>{index + 1}</td>
                <td>{contact.name}</td>
                <td>{contact.phone}</td>
                <td>{contact.email}</td>
                <td>
                  <button className="btn btn-success me-2" onClick={() => handleEditShow(contact)}>Modifier</button>
                  <button className="btn btn-danger" onClick={() => handleDeleteShow(contact)}>Supprimer</button>
                </td>
              </tr>
            )}
          </tbody>
        </Table>
        : 'Aucun contact trouvé'
      </Col>
      <Col lg={4}>
        <h2>Ajouter un contact</h2>
        <Form>
          <Form.Group as={Row} controlId="nom">
            <Form.Label column sm={3}>Nom</Form.Label>
            <Col sm={9}>
              <Form.Control type="text" placeholder="Entrez le nom" />
            </Col>
          </Form.Group>

          <Form.Group as={Row} controlId="phone">
            <Form.Label column sm={3}>Phone</Form.Label>
            <Col sm={9}>
              <Form.Control type="tel" placeholder="Entrez le numéro" />
            </Col>
          </Form.Group>

          <Form.Group as={Row} controlId="email">
            <Form.Label column sm={3}>Email</Form.Label>
            <Col sm={9}>
              <Form.Control type="email" placeholder="Entrez l'email" />
            </Col>
          </Form.Group>

          <Button variant="primary" type="submit" className="mt-3">
            Ajouter
          </Button>
        </Form>
      </Col>
    </Row>
  </Container>

  {/* Modal Modifier */}
  <Modal show={showEdit} onHide={handleEditClose}>

```

```

<Modal.Header closeButton>
  <Modal.Title>Modifier contact</Modal.Title>
</Modal.Header>
<Modal.Body>
  <Form>
    <Form.Group controlId="editNom">
      <Form.Label column sm={3}>Nom</Form.Label>
      <Col sm={9}>
        <Form.Control
          type="text"
          name="name"
          value={formData.name}
          onChange={handleEditChange}
          placeholder="Entrez le nom"
        />
      </Col>
    </Form.Group>

    <Form.Group controlId="editPhone">
      <Form.Label column sm={3}>Phone</Form.Label>
      <Col sm={9}>
        <Form.Control
          type="tel"
          name="phone"
          value={formData.phone}
          onChange={handleEditChange}
          placeholder="Entrez le numéro"
        />
      </Col>
    </Form.Group>

    <Form.Group controlId="editEmail">
      <Form.Label column sm={3}>Email</Form.Label>
      <Col sm={9}>
        <Form.Control
          type="email"
          name="email"
          value={formData.email}
          onChange={handleEditChange}
          placeholder="Entrez l'email"
        />
      </Col>
    </Form.Group>
  </Form>
</Modal.Body>
<Modal.Footer>
  <Button variant="secondary" onClick={handleEditClose}>Fermer</Button>
  <Button variant="primary" onClick={handleEditSave}>Sauvegarder</Button>
</Modal.Footer>
</Modal>

/* Modal Supprimer */
<Modal show={showDelete} onHide={handleDeleteClose}>
  <Modal.Header closeButton>
    <Modal.Title>Supprimer contact</Modal.Title>
  </Modal.Header>
  <Modal.Body>Voulez-vous vraiment supprimer ce contact ?</Modal.Body>
  <Modal.Footer>
    <Button variant="secondary" onClick={handleDeleteClose}>Fermer</Button>
    <Button variant="danger" onClick={handleDeleteConfirm}>Supprimer</Button>
  </Modal.Footer>
</Modal>
</div>
);

}

export default App;

```

16. Ajouter les deux actions de modification et suppression

```

import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css';

```

```

import { Modal, Navbar, Table, Nav, Form, Row, Col, Button, Container } from
'react-bootstrap';
import { useState } from 'react';
import { useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { ListerContacts } from './actions/contact.actions';

function App() {
  const [showEdit, setShowEdit] = useState(false);
  const [showDelete, setShowDelete] = useState(false);
  const [currentContact, setCurrentContact] = useState(null);
  const [formData, setFormData] = useState({ name: '', phone: '', email: '' });

  const handleEditClose = () => setShowEdit(false);
  const handleEditShow = (contact) => {
    setCurrentContact(contact);
    setFormData({ name: contact.name, phone: contact.phone, email: contact.email });
  };
  setShowEdit(true);
};

const handleDeleteClose = () => setShowDelete(false);
const handleDeleteShow = (contact) => {
  setCurrentContact(contact);
  setShowDelete(true);
};

const handleEditChange = (e) => {
  setFormData({ ...formData, [e.target.name]: e.target.value });
};

const handleEditSave = async () => {
  if (currentContact && currentContact._id) {
    try {
      const response = await fetch(`http://localhost:8000/contact/${currentContact._id}/modifier`, {
        method: 'PUT',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(formData),
      });

      if (response.ok) {
        console.log('Contact updated');
        dispatch(ListerContacts());
      } else {
        console.error('Failed to update contact');
      }
    } catch (error) {
      console.error('Error updating contact:', error);
    }
  } else {
    console.error('No contact ID found');
  }
  setShowEdit(false);
};

const handleDeleteConfirm = async () => {
  if (currentContact && currentContact._id) {
    try {
      const response = await fetch(`http://localhost:8000/contact/${currentContact._id}/supprimer`, {
        method: 'DELETE',
      });

      if (response.ok) {
        console.log('Contact deleted');
        dispatch(ListerContacts());
      } else {
        console.error('Failed to delete contact');
      }
    } catch (error) {
  
```

```

        console.error('Error deleting contact:', error);
    }
} else {
    console.error('No contact ID found');
}
setShowDelete(false);
};

const dispatch = useDispatch();
const contacts = useSelector(state => state.contact.contacts);

useEffect(() => {
    dispatch(ListerContacts());
}, [dispatch]);

return (
    <div className="App">
        <Navbar bg="primary" variant="dark">
            <Container>
                <Navbar.Brand href="#home">Contacts Manager</Navbar.Brand>
                <Nav className="me-auto">
                    <Nav.Link href="#home">V2.0</Nav.Link>
                </Nav>
            </Container>
        </Navbar>
        <Container>
            <Row>
                <Col lg={8}>
                    <h2 className="mt-4">Liste des contacts</h2>
                    {contacts ?
                        <Table striped bordered hover>
                            <thead>
                                <tr>
                                    <th>#</th>
                                    <th>Nom</th>
                                    <th>Phone</th>
                                    <th>Email</th>
                                    <th>Actions</th>
                                </tr>
                            </thead>
                            <tbody>
                                {contacts.map((contact, index) =>
                                    <tr key={index}>
                                        <td>{index + 1}</td>
                                        <td>{contact.name}</td>
                                        <td>{contact.phone}</td>
                                        <td>{contact.email}</td>
                                        <td>
                                            <button className="btn btn-success me-2" onClick={() => handleEditShow(contact)}>Modifier</button>
                                            <button className="btn btn-danger" onClick={() => handleDeleteShow(contact)}>Supprimer</button>
                                        </td>
                                    </tr>
                                )
                            )
                        )
                    </tbody>
                </Table>
                : 'Aucun contact trouvé'
            </Col>
            <Col lg={4}>
                <h2>Ajouter un contact</h2>
                <Form>
                    <Form.Group controlId="nom">
                        <Form.Label column sm={3}>Nom</Form.Label>
                        <Col sm={9}>
                            <Form.Control type="text" placeholder="Entrez le nom" />
                        </Col>
                    </Form.Group>

                    <Form.Group controlId="phone">
                        <Form.Label column sm={3}>Phone</Form.Label>
                        <Col sm={9}>
                            <Form.Control type="tel" placeholder="Entrez le numéro" />
                        </Col>
                    </Form.Group>
                </Form>
            </Col>
        </Row>
    </Container>
)
);

```

```

        </Col>
    </Form.Group>

    <Form.Group controlId="email">
        <Form.Label column sm={3}>Email</Form.Label>
        <Col sm={9}>
            <Form.Control type="email" placeholder="Entrez l'email" />
        </Col>
    </Form.Group>

    <Button variant="primary" type="submit" className="mt-3">
        Ajouter
    </Button>
</Form>
</Col>
</Row>
</Container>

{ /* Modal Modifier */
<Modal show={showEdit} onHide={handleEditClose}>
    <Modal.Header closeButton>
        <Modal.Title>Modifier contact</Modal.Title>
    </Modal.Header>
    <Modal.Body>
        <Form>
            <Form.Group controlId="editNom">
                <Form.Label column sm={3}>Nom</Form.Label>
                <Col sm={9}>
                    <Form.Control
                        type="text"
                        name="name"
                        value={formData.name}
                        onChange={handleEditChange}
                        placeholder="Entrez le nom"
                    />
                </Col>
            </Form.Group>

            <Form.Group controlId="editPhone">
                <Form.Label column sm={3}>Phone</Form.Label>
                <Col sm={9}>
                    <Form.Control
                        type="tel"
                        name="phone"
                        value={formData.phone}
                        onChange={handleEditChange}
                        placeholder="Entrez le numéro"
                    />
                </Col>
            </Form.Group>

            <Form.Group controlId="editEmail">
                <Form.Label column sm={3}>Email</Form.Label>
                <Col sm={9}>
                    <Form.Control
                        type="email"
                        name="email"
                        value={formData.email}
                        onChange={handleEditChange}
                        placeholder="Entrez l'email"
                    />
                </Col>
            </Form.Group>

            <Button variant="primary" onClick={handleEditSave} className="mt-3">Enregistrer</Button>
        </Form>
    </Modal.Body>
</Modal>

{ /* Modal Supprimer */
<Modal show={showDelete} onHide={handleDeleteClose}>
    <Modal.Header closeButton>

```

```

<Modal.Title>Supprimer contact</Modal.Title>
</Modal.Header>
<Modal.Body>
  <p>Êtes-vous sûr de vouloir supprimer ce contact ?</p>
  <Button variant="danger" onClick={handleDeleteConfirm}>Confirmer</Button>
</Modal.Body>
</Modal>
</div>
);

}

export default App;

```

La liste des contacts

#	Nom	Tél	Actions	
0	Walid siraji	66585587	<button>Modifier</button>	<button>supprimer</button>
1	kamel abbassi	66585587	<button>Modifier</button>	<button>supprimer</button>
2	Walid chaima	66585587	<button>Modifier</button>	<button>supprimer</button>
3	Ayari chaima	66585587	<button>Modifier</button>	<button>supprimer</button>
4	Ayari chaima	66585587	<button>Modifier</button>	<button>supprimer</button>
5	Sonia ben mourad	66585587	<button>Modifier</button>	<button>supprimer</button>

17. Mise en place du formulaire d'ajout

Ajouter un contact

Nom	<input type="text" value="Entrez le nom"/>
Phone	<input type="text" value="Entrez le numéro"/>
Email	<input type="text" value="Entrez l'email"/>

Ajouter

```

import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import { Modal, Navbar, Table, Nav, Form, Row, Col, Button, Container } from 'react-bootstrap';
import { useState } from 'react';
import { useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { ListerContacts } from './actions/contact.actions';

function App() {
  const [showEdit, setShowEdit] = useState(false);
  const [showDelete, setShowDelete] = useState(false);
  const [currentContact, setCurrentContact] = useState(null);
  const [formData, setFormData] = useState({ name: '', phone: '', email: '' });

  const handleEditClose = () => setShowEdit(false);
  const handleEditShow = (contact) => {
    setCurrentContact(contact);
  }
}

```

```

        setFormData({ name: contact.name, phone: contact.phone, email: contact.email
    });
    setShowEdit(true);
};

const handleDeleteClose = () => setShowDelete(false);
const handleDeleteShow = (contact) => {
    setCurrentContact(contact);
    setShowDelete(true);
};

const handleEditChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
};

const handleEditSave = async () => {
    if (currentContact && currentContact._id) {
        try {
            const response = await fetch(`http://localhost:8000/contact/${currentContact._id}/modifier`, { // Changement du port
                method: 'PUT',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify(formData),
            });

            if (response.ok) {
                console.log('Contact updated');
                dispatch(ListerContacts());
            } else {
                console.error('Failed to update contact');
            }
        } catch (error) {
            console.error('Error updating contact:', error);
        }
    } else {
        console.error('No contact ID found');
    }
    setShowEdit(false);
};

const handleDeleteConfirm = async () => {
    if (currentContact && currentContact._id) {
        try {
            const response = await fetch(`http://localhost:8000/contact/${currentContact._id}/supprimer`, { // Changement du port
                method: 'DELETE',
            });

            if (response.ok) {
                console.log('Contact deleted');
                dispatch(ListerContacts());
            } else {
                console.error('Failed to delete contact');
            }
        } catch (error) {
            console.error('Error deleting contact:', error);
        }
    } else {
        console.error('No contact ID found');
    }
    setShowDelete(false);
};

const handleAddSubmit = async (e) => {
    e.preventDefault();

    try {
        const response = await fetch('http://localhost:8000/contact/ajouter', { // Changement du port

```

```

    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(formData),
  });

  if (response.ok) {
    console.log('Contact added');
    dispatch(ListerContacts());
    // Clear the form after successful submission
    setFormData({ name: '', phone: '', email: '' });
  } else {
    console.error('Failed to add contact');
  }
} catch (error) {
  console.error('Error adding contact:', error);
}
};

const dispatch = useDispatch();
const contacts = useSelector(state => state.contact.contacts);

useEffect(() => {
  dispatch(ListerContacts());
}, [dispatch]);

return (
  <div className="App">
    <Navbar bg="primary" variant="dark">
      <Container>
        <Navbar.Brand href="#home">Contacts Manager</Navbar.Brand>
        <Nav className="me-auto">
          <Nav.Link href="#home">V2.0</Nav.Link>
        </Nav>
      </Container>
    </Navbar>
    <Container>
      <Row>
        <Col lg={8}>
          <h2 className="mt-4">Liste des contacts</h2>
          {contacts ?
            <Table striped bordered hover>
              <thead>
                <tr>
                  <th>#</th>
                  <th>Nom</th>
                  <th>Phone</th>
                  <th>Email</th>
                  <th>Actions</th>
                </tr>
              </thead>
              <tbody>
                {contacts.map((contact, index) =>
                  <tr key={index}>
                    <td>{index + 1}</td>
                    <td>{contact.name}</td>
                    <td>{contact.phone}</td>
                    <td>{contact.email}</td>
                    <td>
                      <button className="btn btn-success me-2" onClick={() => handleEditShow(contact)}>Modifier</button>
                      <button className="btn btn-danger" onClick={() => handleDeleteShow(contact)}>Supprimer</button>
                    </td>
                  </tr>
                )
              )
            </tbody>
          </Table>
          : 'Aucun contact trouvé'
        </Col>
        <Col lg={4}>
          <h2>Ajouter un contact</h2>
        </Col>
      </Row>
    </Container>
  </div>
);

```

```

<Form onSubmit={handleAddSubmit}>
  <Form.Group as={Row} controlId="nom">
    <Form.Label column sm={3}>Nom</Form.Label>
    <Col sm={9}>
      <Form.Control
        type="text"
        placeholder="Entrez le nom"
        value={formData.name}
        onChange={handleEditChange}
        name="name"
      />
    </Col>
  </Form.Group>

  <Form.Group as={Row} controlId="phone">
    <Form.Label column sm={3}>Phone</Form.Label>
    <Col sm={9}>
      <Form.Control
        type="tel"
        placeholder="Entrez le numéro"
        value={formData.phone}
        onChange={handleEditChange}
        name="phone"
      />
    </Col>
  </Form.Group>

  <Form.Group as={Row} controlId="email">
    <Form.Label column sm={3}>Email</Form.Label>
    <Col sm={9}>
      <Form.Control
        type="email"
        placeholder="Entrez l'email"
        value={formData.email}
        onChange={handleEditChange}
        name="email"
      />
    </Col>
  </Form.Group>

  <Button variant="primary" type="submit" className="mt-3">
    Ajouter
  </Button>
</Form>
</Col>
</Row>
</Container>

/* Modal Modifier */
<Modal show={showEdit} onHide={handleEditClose}>
  <Modal.Header closeButton>
    <Modal.Title>Modifier contact</Modal.Title>
  </Modal.Header>
  <Modal.Body>
    <Form>
      <Form.Group as={Row} controlId="editNom">
        <Form.Label column sm={3}>Nom</Form.Label>
        <Col sm={9}>
          <Form.Control
            type="text"
            name="name"
            value={formData.name}
            onChange={handleEditChange}
            placeholder="Entrez le nom"
          />
        </Col>
      </Form.Group>

      <Form.Group as={Row} controlId="editPhone">
        <Form.Label column sm={3}>Phone</Form.Label>
        <Col sm={9}>
          <Form.Control
            type="tel"

```

```

        name="phone"
        value={formData.phone}
        onChange={handleEditChange}
        placeholder="Entrez le numéro"
      />
    </Col>
  </Form.Group>

  <Form.Group controlId="editEmail">
    <Form.Label column sm={3}>Email</Form.Label>
    <Col sm={9}>
      <Form.Control
        type="email"
        name="email"
        value={formData.email}
        onChange={handleEditChange}
        placeholder="Entrez l'email"
      />
    </Col>
  </Form.Group>
</Form>
</Modal.Body>
<Modal.Footer>
  <Button variant="secondary" onClick={handleEditClose}>Fermer</Button>
  <Button variant="primary" onClick={handleEditSave}>Sauvegarder</Button>
</Modal.Footer>
</Modal>

{/* Modal Supprimer */}
<Modal show={showDelete} onHide={handleDeleteClose}>
  <Modal.Header closeButton>
    <Modal.Title>Supprimer contact</Modal.Title>
  </Modal.Header>
  <Modal.Body>Voulez-vous vraiment supprimer ce contact ?</Modal.Body>
  <Modal.Footer>
    <Button variant="secondary" onClick={handleDeleteClose}>Fermer</Button>
    <Button variant="danger" onClick={handleDeleteConfirm}>Supprimer</Button>
  </Modal.Footer>
</Modal>
</div>
);
}

export default App;

```

Erreurs :

Erreur 1:

The href attribute requires a valid value to be accessible. Provide a valid, navigable address as the href value. If you cannot provide a valid href, but still need the element to resemble a link, use a button and change it with appropriate styles. Learn more: <https://github.com>.

Solution 1

Try to replace href="#" with href="#/" inside <a> element, this should fix the problem.

Erreur 2

axios Error: Request failed with status code 400

Solution 2

Ajouter ce code pour afficher le format de données envoyées (data)

```
console.log(error.response.data);
```

Source : <https://dev.to/zelig880/how-to-catch-the-body-of-an-axios-error-4lk0>

