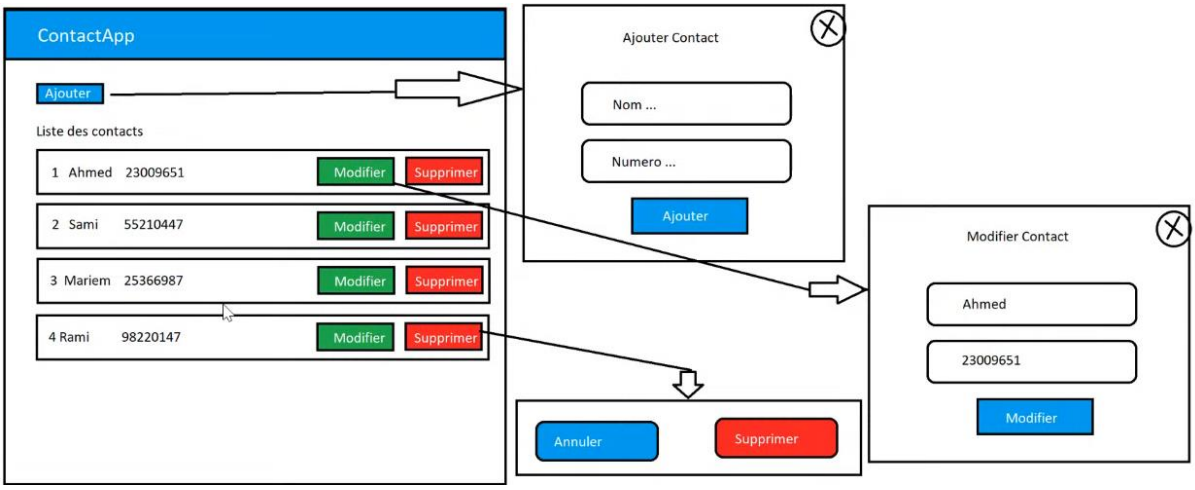
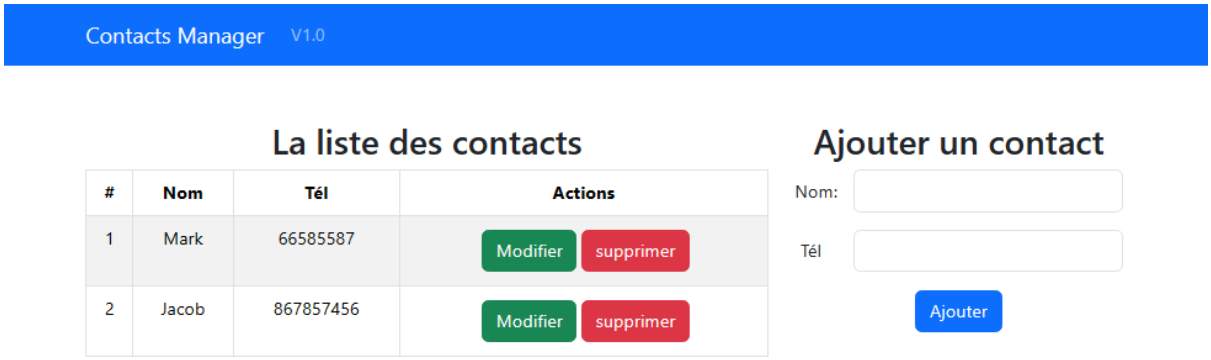


Essayer d'appliquer ce thème dans l'application client (Frontend)



Entête

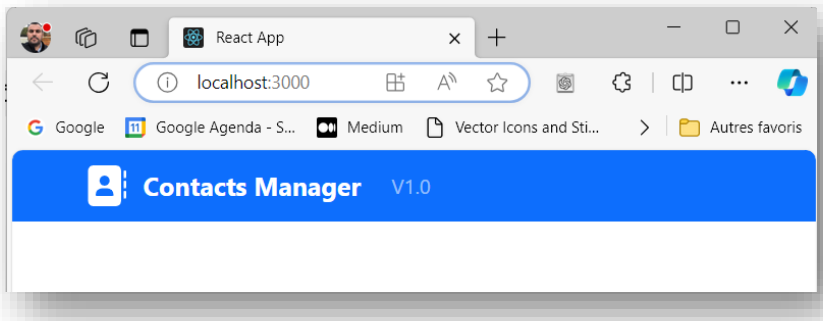
Ajouter fortawesome dans votre projet

```
npm install --save @fortawesome/fontawesome-svg-core @fortawesome/free-solid-svg-icons @fortawesome/react-fontawesome
```

Taper le code suivant dans App.js

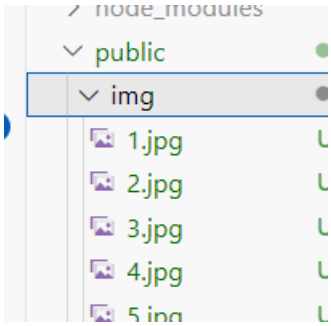
```
import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css';

import { Navbar, Nav, Container } from 'react-bootstrap';

// Importer les packages nécessaires
import { library } from '@fortawesome/fontawesome-svg-core';
import { faAddressBook} from '@fortawesome/free-solid-svg-icons';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import '@fortawesome/fontawesome-svg-core/styles.css';

// Ajouter les icônes à la bibliothèque
library.add(faAddressBook);

function App() {

  return (
    <>
    <div className="App ">
      <Navbar bg="primary" data-bs-theme="dark">
        <Container>
```

```
        <FontAwesomeIcon icon={faAddressBook} size="2x" color="white"/> 
 
          <Navbar.Brand href="#home">Contacts Manager</Navbar.Brand>
          <Nav className="me-auto">
            <Nav.Link href="#home">V1.0</Nav.Link>
          </Nav>
        </Container>
      </Navbar>

    </div>
    </>
  );
}

export default App;
```



Liste des contacts

Créer un dossier **public/img** avec quelques photos des contacts



Créer un composant **ContactsTable** pour afficher la liste des contacts

```
import React from 'react';
import { Table } from 'react-bootstrap';

const ContactsTable = ({ contacts }) => {
  return (
    <Table striped bordered hover>
      <thead>
        <tr>
          <th>Numéro</th>
          <th>Avatar</th>
          <th>Nom</th>
          <th>Téléphone</th>
        </tr>
      </thead>
      <tbody>
        {contacts.map((contact, index) => (
          <tr key={index}>
            <td>{index + 1}</td>
```

```
            <td>
              <img
                src={contact.avatarUrl || 'https://picsum.photos/200/300'}
                alt={`Avatar de ${contact.nom}`}
                width="50"
                height="50"
              />
            </td>
            <td>{contact.nom}</td>
            <td>{contact.tel}</td>
          </tr>
        ))}
      </tbody>
    </Table>
  );
};

export default ContactsTable;
```

Dans le fichier **App.js** modifier le code comme suit :

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

import { Navbar, Nav, Container } from 'react-bootstrap';
import ContactsTable from './components/ContactsTable';

// Importer les packages nécessaires
import '@fortawesome/fontawesome-svg-core/styles.css';
import { library } from '@fortawesome/fontawesome-svg-core';
import { faAddressBook } from '@fortawesome/free-solid-svg-icons';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';

// Ajouter les icônes à la bibliothèque
library.add(faAddressBook);

function App() {

  const contactsData = [
    { nom: "ABBASSI Kamel", tel: 26388202, avatarUrl: "img/6.jpg" },
    { nom: "Haddad faycel", tel: 9874521, avatarUrl: "img/5.jpg" },
    { nom: "Najeh walid", tel: 65465465, avatarUrl: "img/inconnu.jpg" },
    { nom: "Ben said nabiha", tel: 65465465, avatarUrl: "img/7.jpg" }
  ];

  return (
    <>
      <div className="App ">
        <Navbar bg="primary" data-bs-theme="dark">
          <Container>
            <FontAwesomeIcon icon={faAddressBook} size="2x" color="white"
/>    
            <Navbar.Brand href="#home">Contacts Manager</Navbar.Brand>
            <Nav className="me-auto">
              <Nav.Link href="#home">V1.0</Nav.Link>
            </Nav>
          </Container>
        </Navbar>

        <div className='container'>
          <div className='p-4'></div>
          <ContactsTable contacts={contactsData} />
```

```
        </div>

      </div>
    </>
  );
}

export default App;
```

Ajouter un système de pagination

```
//ContactsTable.js

import React ,{ useState }from 'react';
import { Table, Pagination , Badge } from 'react-bootstrap';

const ContactsTable = ({ contacts }) => {

    //Systeme de pagination
    const [currentPage, setCurrentPage] = useState(1);
    const contactsPerPage = 5; // Nombre de contacts par page

    if (!contacts || contacts.length === 0) {
      return <p>No contacts available.</p>;
    }

    // Calcule l'index du premier et du dernier contact sur la page actuelle
    const indexOfLastContact = currentPage * contactsPerPage;
    const indexOfFirstContact = indexOfLastContact - contactsPerPage;
    const currentContacts = contacts.slice(indexOfFirstContact,
indexOfLastContact);

    const totalPages = Math.ceil(contacts.length / contactsPerPage);

    const paginate = (pageNumber) => setCurrentPage(pageNumber);
    //Fin systeme de pagination

  return (
    <>
  <h2 className="text-muted">Votre liste des contacts contient <Badge
variant="primary">{contacts.length}</Badge> contacts</h2>

    <Table striped bordered hover>
      <thead>
        <tr>
          <th>Numéro</th>
          <th>Avatar</th>
          <th>Nom</th>
          <th>Téléphone</th>
        </tr>
      </thead>
      <tbody>
        {currentContacts.map((contact, index) => (
          <tr key={index}>
             <td>{indexOfFirstContact + index + 1}</td>
            <td>
              <img
                src={contact.avatarUrl || 'https://picsum.photos/200/300'}
                alt={`Avatar de ${contact.nom}`}
                width="50"
                height="50"
              />
            </td>
            <td>{contact.nom}</td>
            <td>{contact.tel}</td>
          </tr>
        ))}
      </tbody>
    </Table>
    <Pagination>
```

```
        {Array.from({ length: totalPages }).map((_, index) => (
          <Pagination.Item
            key={index + 1}
            active={index + 1 === currentPage}
            onClick={() => paginate(index + 1)}
          >
            {index + 1}
          </Pagination.Item>
        ))}
      </Pagination>
    </>
  );
};

export default ContactsTable;
```

Ajouter dans App.js un ensemble des constats :

```
const contactsData = [
    { nom: "ABBASSI Kamel", tel: 26388202, avatarUrl: "img/6.jpg" },
    { nom: "Haddad faycel", tel: 9874521, avatarUrl: "img/5.jpg" },
    { nom: "Najeh walid", tel: 65465465, avatarUrl: "img/inconnu.jpg" },
    { nom: "Ben said nabiha", tel: 65465465, avatarUrl: "img/7.jpg" },
    { nom: "Guesmi Samir", tel: 65465465, avatarUrl: "img/inconnu.jpg" },
    { nom: "Gattoussi Ali", tel: 65465465, avatarUrl: "img/7.jpg" },
    { nom: "Taieb Dalinda", tel: 65465465, avatarUrl: "img/4.jpg" },
    { nom: "Ben achour sameh", tel: 65465465, avatarUrl: "img/6.jpg" }
  ];
```

Explication

Ce code représente un composant `ContactsTable` en React, qui affiche une table de contacts avec un système de pagination. Voici une explication détaillée du code :

1. Importations :

```
    import React, { useState } from 'react';
    import { Table, Pagination } from 'react-bootstrap';
```

- `React` est nécessaire pour définir des composants React.

- `useState` est un hook React qui permet d'ajouter un état local à un composant fonctionnel.

- `Table` et `Pagination` sont des composants de l'ensemble React Bootstrap, utilisés pour afficher la table des contacts et le système de pagination.

2. Déclaration du composant `ContactsTable` :

```
    const ContactsTable = ({ contacts }) => {
```

- Le composant prend un seul argument, `contacts`, qui est l'ensemble des contacts à afficher.

3. Système de Pagination :

```
    const [currentPage, setCurrentPage] = useState(1);
    const contactsPerPage = 5; // Nombre de contacts par page

    if (!contacts || contacts.length === 0) {
      return <p>No contacts available.</p>;
    }

    // Calcule l'index du premier et du dernier contact sur la page actuelle
    const indexOfLastContact = currentPage * contactsPerPage;
    const indexOfFirstContact = indexOfLastContact - contactsPerPage;
    const currentContacts = contacts.slice(indexOfFirstContact,
indexOfLastContact);

    const totalPages = Math.ceil(contacts.length / contactsPerPage);
```

```
    const paginate = (pageNumber) => setCurrentPage(pageNumber);
```

- Utilisation du hook `useState` pour définir l'état `currentPage` qui représente la page actuelle.

- `contactsPerPage` détermine le nombre de contacts à afficher par page.

- Vérification si les contacts sont disponibles, sinon, un message "No contacts available." est affiché.

- Calcul des index du premier et du dernier contact sur la page actuelle.

- `currentContacts` contient les contacts à afficher sur la page actuelle.

- Calcul du nombre total de pages (`totalPages`) en fonction du nombre total de contacts et du nombre de contacts par page.

- La fonction `paginate` permet de changer la page en mettant à jour l'état `currentPage`.

4. Affichage de la Table des Contacts :

```
  return (
    <>
    <Table striped bordered hover>
      {/* ... en-tête du tableau */}
      <tbody>
        {currentContacts.map((contact, index) => (
          <tr key={index}>
             <td>{indexOfFirstContact + index + 1}</td>
            <td>
              <img
                src={contact.avatarUrl || 'https://picsum.photos/200/300'}
                alt={`Avatar de ${contact.nom}`}
                width="50"
                height="50"
              />
            </td>
            <td>{contact.nom}</td>
            <td>{contact.tel}</td>
          </tr>
        ))}
      </tbody>
    </Table>
    <Pagination>
        {Array.from({ length: totalPages }).map((_, index) => (
          <Pagination.Item
            key={index + 1}
            active={index + 1 === currentPage}
            onClick={() => paginate(index + 1)}
          >
            {index + 1}
          </Pagination.Item>
        ))}
      </Pagination>
   </>
  );
```

- Affichage de la table des contacts avec les données des contacts actuels (`currentContacts`).

- Utilisation de la fonction `map` pour itérer sur les contacts et générer les lignes de la table.

- Affichage de l'image avec une URL spécifiée dans `avatarUrl` ou une image de remplacement (`https://picsum.photos/200/300` si `avatarUrl` est vide).

- Affichage du système de pagination avec des boutons pour chaque page. L'option `active` indique la page actuelle.

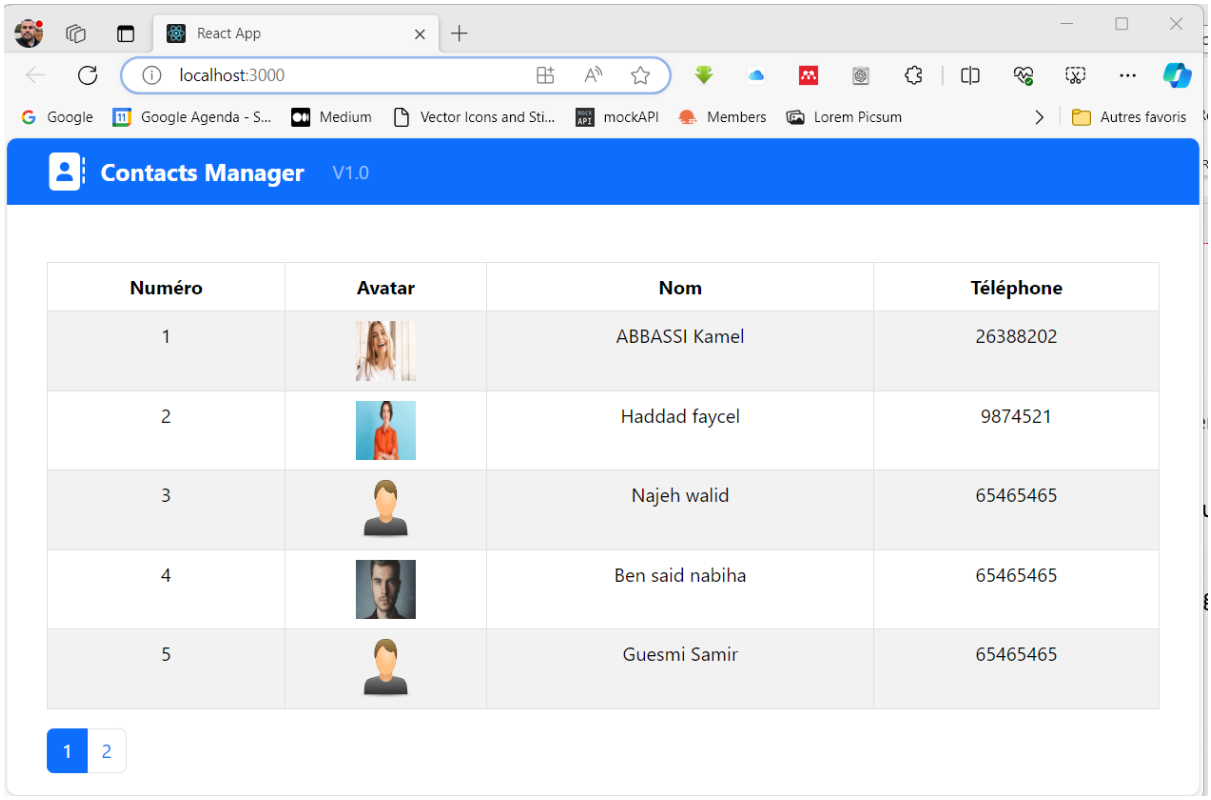
- Un clic sur un bouton de pagination appelle la fonction `paginate` pour mettre à jour la page actuelle.

Ajouter un système de tri

```
import React, { useState } from 'react';
import { Table, Pagination, Badge } from 'react-bootstrap';

const ContactsTable = ({ contacts }) => {
  const [currentPage, setCurrentPage] = useState(1);
  const [sortBy, setSortBy] = useState(null);
  const [sortOrder, setSortOrder] = useState('asc');
  const contactsPerPage = 5;

  if (!contacts || contacts.length === 0) {
    return <p>No contacts available.</p>;
  }

  const indexOfLastContact = currentPage * contactsPerPage;
  const indexOfFirstContact = indexOfLastContact - contactsPerPage;
  const sortedContacts = contacts.slice().sort((a, b) => {
    if (sortBy === 'nom') {
      return sortOrder === 'asc' ? a.nom.localeCompare(b.nom) :
b.nom.localeCompare(a.nom);
    }
    // Ajoutez d'autres critères de tri si nécessaire
    return 0;
  });

  const currentContacts = sortedContacts.slice(indexOfFirstContact,
indexOfLastContact);
  const totalPages = Math.ceil(sortedContacts.length / contactsPerPage);

  const paginate = (pageNumber) => setCurrentPage(pageNumber);
  const handleSort = (key) => {
    if (sortBy === key) {
      setSortOrder(sortOrder === 'asc' ? 'desc' : 'asc');
    } else {
      setSortBy(key);
      setSortOrder('asc');
    }
  };

  return (
    <>
      <h2 className="text-muted">
        Votre liste des contacts contient{' '}
        <Badge variant="primary">{contacts.length}</Badge> contacts
      </h2>
<p>
contacts triés par: <span class="badge bg-primary">{sortBy ? "" + sortBy :
""}</span>
      </p>
      <Table striped bordered hover>
        <thead>
          <tr>
            <th onClick={() => handleSort('numero')}>Numéro</th>
            <th>Avatar</th>
            <th onClick={() => handleSort('nom')}>Nom</th>
            <th onClick={() => handleSort('tel')}>Téléphone</th>
          </tr>
        </thead>
        <tbody>
          {currentContacts.map((contact, index) => (
            <tr key={index}>
              <td>{indexOfFirstContact + index + 1}</td>
              <td>
```

```
                <img
                  src={contact.avatarUrl || 'https://picsum.photos/200/300'}
                  alt={`Avatar de ${contact.nom}`}
                  width="50"
                  height="50"
                />
              </td>
              <td>{contact.nom}</td>
              <td>{contact.tel}</td>
            </tr>
          ))}
        </tbody>
      </Table>
      <Pagination>
        {Array.from({ length: totalPages }).map((_, index) => (
          <Pagination.Item
            key={index + 1}
            active={index + 1 === currentPage}
            onClick={() => paginate(index + 1)}
          >
            {index + 1}
          </Pagination.Item>
        ))}
      </Pagination>
    </>
  );
};

export default ContactsTable;
```

Aperçu



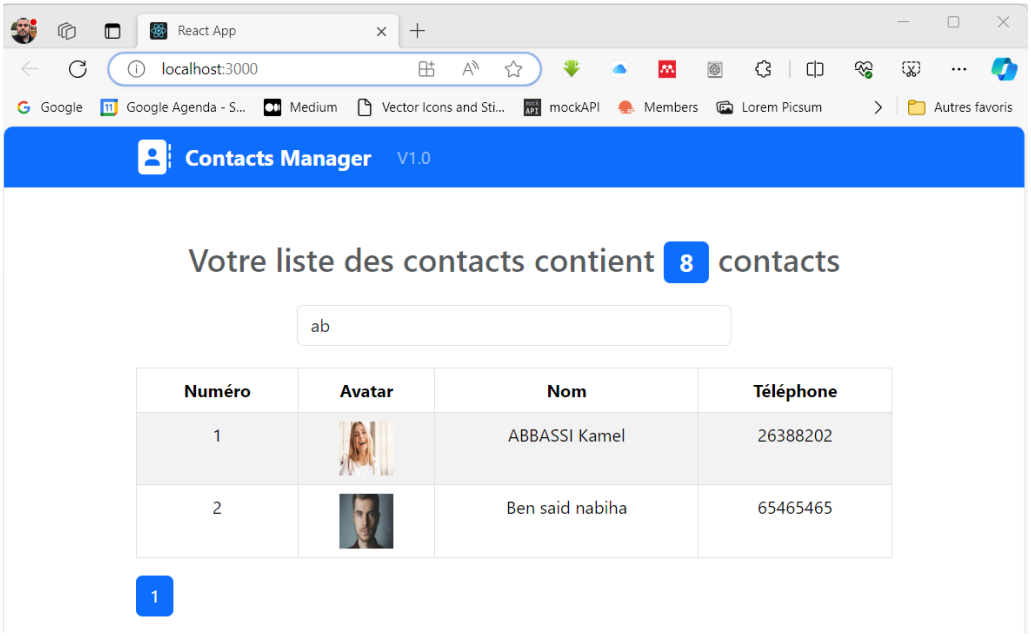
Ajouter un formulaire de recherche par *nom*

```
  //ContactsTable.js
import React, { useState } from 'react';
import { Table, Pagination, Badge, Form } from 'react-bootstrap';

const ContactsTable = ({ contacts }) => {
  const [currentPage, setCurrentPage] = useState(1);
```

```
  const [sortBy, setSortBy] = useState(null);
  const [sortOrder, setSortOrder] = useState('asc');
  const [searchTerm, setSearchTerm] = useState('');
  const contactsPerPage = 5;

  if (!contacts || contacts.length === 0) {
    return <p>No contacts available.</p>;
  }

  const indexOfLastContact = currentPage * contactsPerPage;
  const indexOfFirstContact = indexOfLastContact - contactsPerPage;
  const sortedContacts = contacts.filter(
    (contact) =>
      contact.nom.toLowerCase().includes(searchTerm.toLowerCase()) ||
      contact.tel.toString().includes(searchTerm)
  ).sort((a, b) => {
      if (sortBy === 'numero') {
        return sortOrder === 'asc' ? a.numero - b.numero : b.numero - a.numero;
      } else if (sortBy === 'nom') {
        return sortOrder === 'asc' ? a.nom.localeCompare(b.nom) :
b.nom.localeCompare(a.nom);
      } else if (sortBy === 'tel') {
        return sortOrder === 'asc' ? a.tel - b.tel : b.tel - a.tel;
      }
      // Ajoutez d'autres critères de tri si nécessaire
      return 0;
    });

  const currentContacts = sortedContacts.slice(indexOfFirstContact,
indexOfLastContact);
  const totalPages = Math.ceil(sortedContacts.length / contactsPerPage);

  const paginate = (pageNumber) => setCurrentPage(pageNumber);
  const handleSort = (key) => {
    if (sortBy === key) {
      setSortOrder(sortOrder === 'asc' ? 'desc' : 'asc');
    } else {
      setSortBy(key);
      setSortOrder('asc');
    }
  };

  //Mettre en forme le terme cherché
  const highlightText = (text, query) => {
    if (typeof text !== 'string') {
      text = String(text);
    }
    const parts = text.split(new RegExp(`(${query})`, 'gi'));
    return parts.map((part, i) =>
      part.toLowerCase() === query.toLowerCase() ? <strong key={i}>{part}</strong>
: part
    );
  };

  return (
    <>
      <h2 className="text-muted">
        Votre liste des contacts contient{' '}
        <Badge variant="primary">{contacts.length}</Badge> contacts
      </h2>

      <p>
      contacts triés par: <span class="badge bg-primary">{sortBy ? "" + sortBy :
""}</span>
      </p>

      <Form>
        <Form.Group controlId="searchTerm">
          <Form.Control
            type="text"
            placeholder="Rechercher par nom"
            value={searchTerm}
            onChange={(e) => setSearchTerm(e.target.value)}
```

```
          />
        </Form.Group>
      </Form>
      <div className='m-2'></div>

      <Table striped bordered hover>
        <thead>
          <tr>
            <th onClick={() => handleSort('numero')}>Numéro</th>
            <th>Avatar</th>
            <th onClick={() => handleSort('nom')}>Nom</th>
            <th onClick={() => handleSort('tel')}>Téléphone</th>
          </tr>
        </thead>
        <tbody>
          {currentContacts.map((contact, index) => (
            <tr key={index}>
              <td>{indexOfFirstContact + index + 1}</td>
              <td>
                <img
                  src={contact.avatarUrl || 'https://picsum.photos/200/300'}
                  alt={`Avatar de ${contact.nom}`}
                  width="50"
                  height="50"
                />
              </td>
              <td>
                {highlightText(contact.nom, searchTerm)}
              </td>
              <td>
                {highlightText(contact.tel, searchTerm)}
              </td>
            </tr>
          ))}
        </tbody>
      </Table>

      <Pagination>
        {Array.from({ length: totalPages }).map((_, index) => (
          <Pagination.Item
            key={index + 1}
            active={index + 1 === currentPage}
            onClick={() => paginate(index + 1)}
          >
            {index + 1}
          </Pagination.Item>
        ))}
      </Pagination>
    </>
  );
};

export default ContactsTable;
```

Aperçu



Ajouter un bouton pour exporter la liste des contacts en fichier MS -Excel

Pour exporter les contacts en fichier Excel, vous pouvez utiliser une bibliothèque comme xlsx en conjonction avec un bouton d'export. Voici comment vous pouvez le faire :

1. Installez la bibliothèque xlsx si vous ne l'avez pas encore fait :

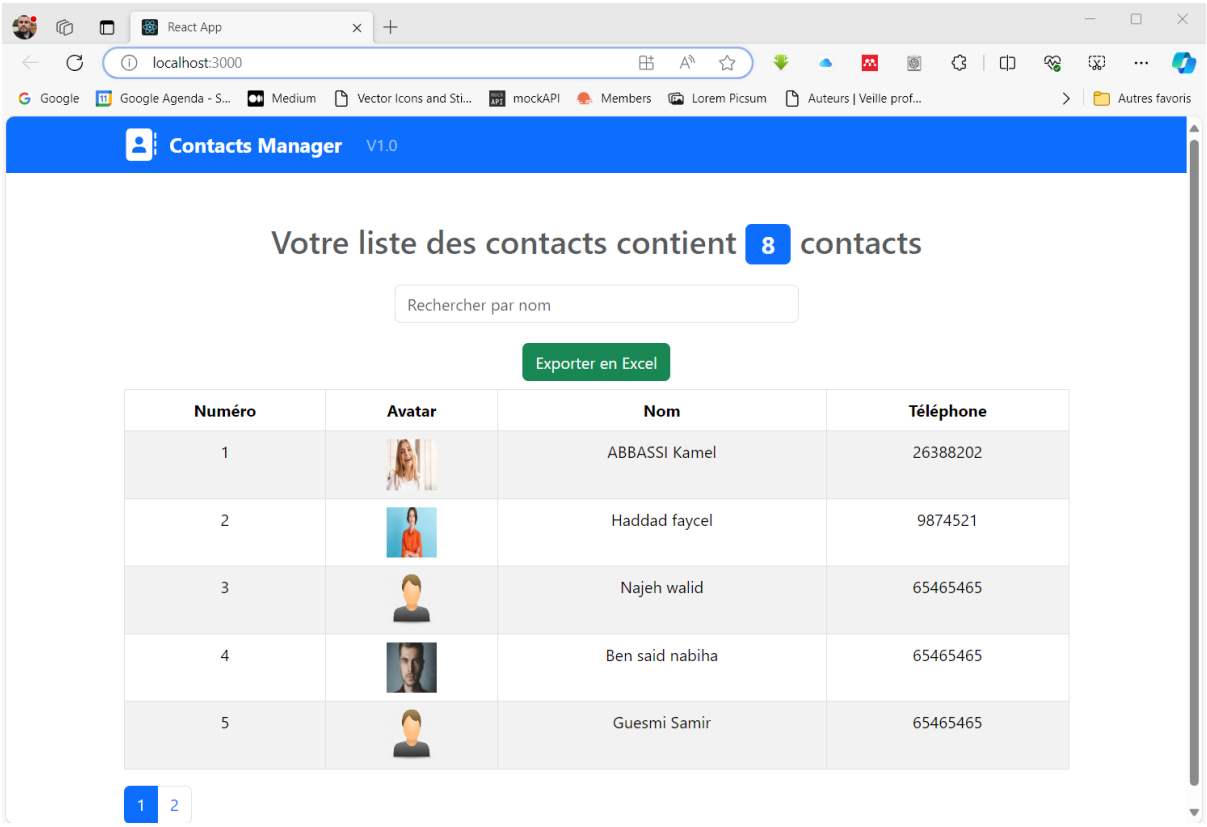
```
npm install xlsx
```

2. Importez xlsx dans votre composant ContactsTable :

```
import React, { useState } from 'react';
import { Table, Pagination, Badge, Form, Button } from 'react-bootstrap';
import * as XLSX from 'xlsx';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faFileExcel } from '@fortawesome/free-solid-svg-icons';

// ... (le reste de votre import)

const ContactsTable = ({ contacts }) => {
  // ... (le reste de votre code)

  const handleExportExcel = () => {
    const dataToExport = sortedContacts.map((contact) => ({
      Numéro: indexOfFirstContact + sortedContacts.indexOf(contact) + 1,
      Avatar: contact.avatarUrl || 'https://picsum.photos/200/300',
      Nom: contact.nom,
      Téléphone: contact.tel,
    }));

    const ws = XLSX.utils.json_to_sheet(dataToExport);
    const wb = XLSX.utils.book_new();
    XLSX.utils.book_append_sheet(wb, ws, 'Contacts');
    XLSX.writeFile(wb, 'contacts.xlsx');
  };

  return (
    <>
      <h2 className="text-muted">
        Votre liste des contacts contient{' '}
        <Badge variant="primary">{contacts.length}</Badge> contacts
      </h2>

      <Form>
        {/* ... (votre champ de recherche) */}
      </Form>
```

```
      <Button variant="success" onClick={handleExportExcel}>
        <FontAwesomeIcon icon={faFileExcel} className="mr-1" /> Exporter en Excel
      </Button>

      <Table striped bordered hover>
        {/* ... (votre tableau) */}
      </Table>

      <Pagination>
        {/* ... (votre pagination) */}
      </Pagination>
    </>
  );
};

export default ContactsTable;
```

Aperçu



Ajouter un bouton pour imprimer **PDF** les contacts visibles

Installer jspdf : **npm i jspdf jspdf-autotable**

```
//ContactsTable.js
import React, { useState } from 'react';
import { Table, Pagination, Badge, Form, Button } from 'react-bootstrap';
import * as XLSX from 'xlsx';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faFileExcel , faFilePdf} from '@fortawesome/free-solid-svg-icons';

// JsPDF
import jsPDF from 'jspdf';
import 'jspdf-autotable';

const ContactsTable = ({ contacts }) => {
  const [currentPage, setCurrentPage] = useState(1);
  const [sortBy, setSortBy] = useState(null);
  const [sortOrder, setSortOrder] = useState('asc');
  const [searchTerm, setSearchTerm] = useState('');
```

```
  const contactsPerPage = 5;

  if (!contacts || contacts.length === 0) {
    return <p>No contacts available.</p>;
  }

  const indexOfLastContact = currentPage * contactsPerPage;
  const indexOfFirstContact = indexOfLastContact - contactsPerPage;
  const sortedContacts = contacts.filter(
    (contact) =>
      contact.nom.toLowerCase().includes(searchTerm.toLowerCase()) ||
      contact.tel.toString().includes(searchTerm)
  ).sort((a, b) => {
    if (sortBy === 'numero') {
      return sortOrder === 'asc' ? a.numero - b.numero : b.numero - a.numero;
    } else if (sortBy === 'nom') {
      return sortOrder === 'asc' ? a.nom.localeCompare(b.nom) :
b.nom.localeCompare(a.nom);
    } else if (sortBy === 'tel') {
      return sortOrder === 'asc' ? a.tel - b.tel : b.tel - a.tel;
    }
    // Ajoutez d'autres critères de tri si nécessaire
    return 0;
  });

  const currentContacts = sortedContacts.slice(indexOfFirstContact,
indexOfLastContact);
  const totalPages = Math.ceil(sortedContacts.length / contactsPerPage);

  const paginate = (pageNumber) => setCurrentPage(pageNumber);
  const handleSort = (key) => {
    if (sortBy === key) {
      setSortOrder(sortOrder === 'asc' ? 'desc' : 'asc');
    } else {
      setSortBy(key);
      setSortOrder('asc');
    }
  };

  //Mettre en forme le terme cherché
  const highlightText = (text, query) => {
    if (typeof text !== 'string') {
      text = String(text);
    }
    const parts = text.split(new RegExp(`(${query})`, 'gi'));
    return parts.map((part, i) =>
      part.toLowerCase() === query.toLowerCase() ? <strong key={i}>{part}</strong>
: part
    );
  };

  const handleExportExcel = () => {
    const dataToExport = sortedContacts.map((contact) => ({
      Numéro: indexOfFirstContact + sortedContacts.indexOf(contact) + 1,
      Avatar: contact.avatarUrl || 'https://picsum.photos/200/300',
      Nom: contact.nom,
      Téléphone: contact.tel,
    }));

    const ws = XLSX.utils.json_to_sheet(dataToExport);
    const wb = XLSX.utils.book_new();
    XLSX.utils.book_append_sheet(wb, ws, 'Contacts');
    XLSX.writeFile(wb, 'contacts.xlsx');
  };

// Function to handle exporting to PDF
const handleExportPDF = () => {
  // Create a new jsPDF instance
  const doc = new jsPDF();

  // Set text color to red
  doc.setTextColor(255, 0, 0);
```

```
  // Set font style to bold
  doc.setFont('bold');

  // Add title in red and bold
  doc.setFontSize(20);
  doc.text('Liste des contacts', 100, 10, { align: 'center' });

  // Reset text color and font style
  doc.setTextColor(0, 0, 0);
  doc.setFont('normal');

  // Add date and time at the top of the page
  const today = new Date();
  const newdat = "Imprimé le: " + today.toLocaleString();
  doc.text(newdat, 10, 20);

  // Add an image at a specific location on the page
  var img = new Image();
  img.src = 'img/elife.jpg';
  doc.addImage(img, 'png', 150, 10, 20, 15);

  // Add some space before the table
  const spaceBeforeTable = 30;

  doc.autoTable({
    head: [['Numéro', 'Avatar', 'Nom', 'Téléphone']],
    body: contacts.map((contact, index) => [
      indexOfFirstContact + index + 1,
      { image: contact.avatarUrl || 'https://picsum.photos/200/300', width: 20,
height: 20 },
      contact.nom,
      contact.tel,
    ]),
    startY: spaceBeforeTable, // Adjust the Y-coordinate to add space

    margin: { top: 10 }, // Add extra margin to accommodate the header
  });
// Add page numbering on all pages
  const pageCount = doc.internal.getNumberOfPages();
  for (let i = 1; i <= pageCount; i++) {
    doc.setPage(i);
    doc.text( 'Page '+ i + '/' + pageCount, 180, doc.internal.pageSize.height -
10);
  }
  // Save the PDF with the specified filename
  doc.save('contacts.pdf');
};

// Note: Make sure to adjust the paths and styling based on your specific
requirements and project structure.



  return (
    <>
      <h2 className="text-muted">
        Votre liste des contacts contient{' '}
        <Badge variant="primary">{contacts.length}</Badge> contacts
      </h2>

      <p>
        contacts triés par: <span class="badge bg-primary">{sortBy ? "" + sortBy :
""}</span>
      </p>

      <Form>
        <Form.Group controlId="searchTerm">
          <Form.Control
            type="text"
            placeholder="Rechercher par nom"
            value={searchTerm}
            onChange={(e) => setSearchTerm(e.target.value)}
          />
```

```
        </Form.Group>
      </Form>
      <div className='m-2'></div>
      <Button variant="success" onClick={handleExportExcel}>
        <FontAwesomeIcon icon={faFileExcel} className="mr-1" /> Exporter en Excel
      </Button>
       

      <Button variant="primary" onClick={handleExportPDF} className="ml-2">
        <FontAwesomeIcon icon={faFilePdf} className="mr-1" /> Exporter en PDF
      </Button>
      <div className='m-2'></div>
      <Table striped bordered hover>
        <thead>
          <tr>
            <th onClick={() => handleSort('numero')}>Numéro</th>
            <th>Avatar</th>
            <th onClick={() => handleSort('nom')}>Nom</th>
            <th onClick={() => handleSort('tel')}>Téléphone</th>
          </tr>
        </thead>
        <tbody>
          {currentContacts.map((contact, index) => (
            <tr key={index}>
              <td>{indexOfFirstContact + index + 1}</td>
              <td>
                <img
                  src={contact.avatarUrl || 'https://picsum.photos/200/300'}
                  alt={`Avatar de ${contact.nom}`}
                  width="50"
                  height="50"
                />
              </td>
              <td>
                {highlightText(contact.nom, searchTerm)}
              </td>
              <td>
                {highlightText(contact.tel, searchTerm)}
              </td>
            </tr>
          ))}
        </tbody>
      </Table>

      <Pagination>
        {Array.from({ length: totalPages }).map((_, index) => (
          <Pagination.Item
            key={index + 1}
            active={index + 1 === currentPage}
            onClick={() => paginate(index + 1)}
          >
            {index + 1}
          </Pagination.Item>
        ))}
      </Pagination>
    </>
  );
};

export default ContactsTable;
```

Modifier le code de fichier App.js pour afficher la liste des contacts de votre API : http://localhost:3001/contact/lister

```
import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import React, { useState, useEffect } from 'react';

import { Navbar, Nav, Container } from 'react-bootstrap';
import ContactsTable from './components/ContactsTable';
```

```
// Importer les packages nécessaires
import '@fortawesome/fontawesome-svg-core/styles.css';
import { library } from '@fortawesome/fontawesome-svg-core';
import { faAddressBook } from '@fortawesome/free-solid-svg-icons';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';

// Ajouter les icônes à la bibliothèque
library.add(faAddressBook);

function App() {

  const [contactsData, setContactsData] = useState([]);

  useEffect(() => {
    const fetchContacts = async () => {
      try {
        const response = await fetch('http://localhost:3001/contact/lister');
        if (!response.ok) {
          throw new Error(`Erreur lors de la récupération des contacts:
${response.statusText}`);
        }
        const data = await response.json();
        // Suppose que la liste des contacts est dans un objet nommé "liste"
        setContactsData(data.liste || []);
      } catch (error) {
        console.error(error.message);
      }
    };

    fetchContacts();
  }, []);


  return (
    <>
      <div className="App ">
        <Navbar bg="primary" data-bs-theme="dark">
          <Container>
            <FontAwesomeIcon icon={faAddressBook} size="2x" color="white"
/>    
            <Navbar.Brand href="#home">Contacts Manager</Navbar.Brand>
            <Nav className="me-auto">
              <Nav.Link href="#home">V1.0</Nav.Link>
            </Nav>
          </Container>
        </Navbar>

        <div className='container'>
          <div className='p-4'></div>
          <ContactsTable contacts={contactsData} />
        </div>

      </div>
    </>
  );
}

export default App;
```

Ajouter un bouton pour afficher un formulaire d'ajout

Ajouter le composant suivant :

```
//AddContactModal.js
import React, { useState } from 'react';
import { Modal, Button, Form } from 'react-bootstrap';

const AddContactModal = ({ show, handleClose, handleAddContact }) => {
```

```
  const [nom, setNom] = useState('');
  const [tel, setTel] = useState('');

  const handleSave = () => {
    // Validez les données si nécessaire
    // Appel de la fonction pour ajouter le contact
    handleAddContact({ nom, tel });

    // Réinitialisez les champs du formulaire
    setNom('');
    setTel('');

    // Fermez le modal
    handleClose();
  };

  return (
    <Modal show={show} onHide={handleClose}>
      <Modal.Header closeButton>
        <Modal.Title>Ajouter un contact</Modal.Title>
      </Modal.Header>
      <Modal.Body>
        <Form>
          <Form.Group controlId="nom">
            <Form.Label>Nom</Form.Label>
            <Form.Control
              type="text"
              placeholder="Entrez le nom"
              value={nom}
              onChange={(e) => setNom(e.target.value)}
            />
          </Form.Group>
          <Form.Group controlId="tel">
            <Form.Label>Téléphone</Form.Label>
            <Form.Control
              type="text"
              placeholder="Entrez le numéro de téléphone"
              value={tel}
              onChange={(e) => setTel(e.target.value)}
            />
          </Form.Group>
        </Form>
      </Modal.Body>
      <Modal.Footer>
        <Button variant="secondary" onClick={handleClose}>
          Annuler
        </Button>
        <Button variant="primary" onClick={handleSave}>
          Enregistrer
        </Button>
      </Modal.Footer>
    </Modal>
  );
};

export default AddContactModal;
```

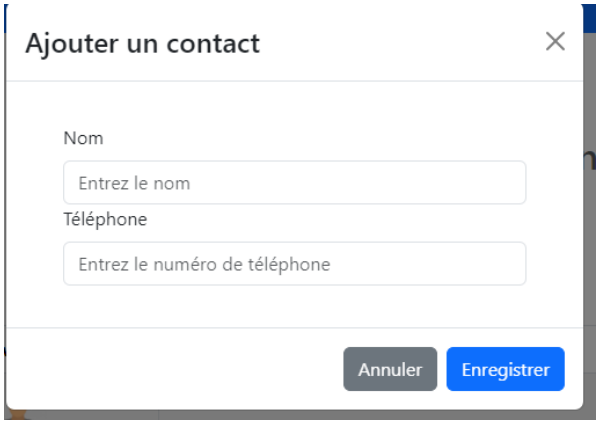
Modifier le fichier **App.js** :

```
 //App.js
import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import React, { useState, useEffect } from 'react';
import { Navbar, Nav, Container ,Button} from 'react-bootstrap';
import ContactsTable from './components/ContactsTable';
import AddContactModal from './components/AddContactModal';
// Importer les packages nécessaires
import '@fortawesome/fontawesome-svg-core/styles.css';
import { library } from '@fortawesome/fontawesome-svg-core';
import { faAddressBook } from '@fortawesome/free-solid-svg-icons';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
```

```
// Ajouter les icônes à la bibliothèque
library.add(faAddressBook);

function App() {
  const [showAddContactModal, setShowAddContactModal] = useState(false);
  const [contactsData, setContactsData] = useState([]);
  const fetchContacts = async () => {
    try {
      const response = await fetch('http://localhost:3001/contact/lister');
      if (!response.ok) {
        throw new Error(`Erreur lors de la récupération des contacts:
${response.statusText}`);
      }
      const data = await response.json();
      // Suppose que la liste des contacts est dans un objet nommé "liste"
      setContactsData(data.liste || []);
    } catch (error) {
      console.error(error.message);
    }
  };

  useEffect(() => {
    fetchContacts();
  }, []);


  const handleAddContact = async (newContact) => {
    try {
      const response = await fetch('http://localhost:3001/contact/ajouter', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(newContact),
      });

      if (response.ok) {
        // La requête a réussi, mettez à jour votre liste de contacts
        fetchContacts(); // Actualisez la liste des contacts après l'ajout
        console.log('Contact ajouté avec succès.');
      } else {
        // La requête a échoué, traitez les erreurs si nécessaire
        console.error('Erreur lors de l\'ajout du contact.');
      }
    } catch (error) {
      console.error('Erreur lors de la requête d\'ajout de contact :', error);
    }

    // Fermez le modal après l'ajout du contact, si nécessaire
    setShowAddContactModal(false);
  };




  return (
    <>
      <div className="App ">
        <Navbar bg="primary" data-bs-theme="dark">
          <Container>
            <FontAwesomeIcon icon={faAddressBook} size="2x" color="white"
/>    
            <Navbar.Brand href="#home">Contacts Manager</Navbar.Brand>
            <Nav className="me-auto">
              <Nav.Link href="#home">V1.0</Nav.Link>
            </Nav>
          </Container>
        </Navbar>

        <div className='container'>
          <div className='p-4'></div>
```

```
          <Button variant="danger" onClick={() => setShowAddContactModal(true)}>
            Ajouter un contact
          </Button>

          <AddContactModal
            show={showAddContactModal}
            handleClose={() => setShowAddContactModal(false)}
            handleAddContact={handleAddContact}
          />

          <ContactsTable contacts={contactsData} />
        </div>

      </div>
    </>
  );
}

export default App;
```

Aperçu



Ajouter des notifications

Installer le package : react-toastify

```
npm install react-toastify
```

Modifier app.js

```
import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import React, { useState, useEffect } from 'react';
import AddContactModal from './components/AddContactModal';
import ContactsTable from './components/ContactsTable';
import { Navbar, Nav, Container, Button } from 'react-bootstrap';
import '@fortawesome/fontawesome-svg-core/styles.css';
import { library } from '@fortawesome/fontawesome-svg-core';
import { faAddressBook } from '@fortawesome/free-solid-svg-icons';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';

library.add(faAddressBook);

function App() {
  const [contactsData, setContactsData] = useState([]);
  const [showAddContactModal, setShowAddContactModal] = useState(false);

  const fetchContacts = async () => {
    try {
      const response = await fetch('http://localhost:3001/contact/lister');
      if (!response.ok) {
```

```
        throw new Error(`Erreur lors de la récupération des contacts:
${response.statusText}`);
      }
      const data = await response.json();
      // Suppose que la liste des contacts est dans un objet nommé "liste"
      setContactsData(data.liste || []);
    } catch (error) {
      console.error(error.message);
    }
  };

  useEffect(() => {
    fetchContacts();
  }, []);

  const handleAddContact = async (newContact) => {
    try {
      const response = await fetch('http://localhost:3001/contact/ajouter', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(newContact),
      });

      if (response.ok) {
        // La requête a réussi, mettez à jour votre liste de contacts
        fetchContacts(); // Actualisez la liste des contacts après l'ajout
        toast.success('Contact ajouté avec succès.', {
          position: 'top-right',
          autoClose: 3000, // Duration in milliseconds
          hideProgressBar: false,
          closeOnClick: true,
          pauseOnHover: true,
          draggable: true,
          progress: undefined,
        });
      } else {
        // La requête a échoué, traitez les erreurs si nécessaire
        toast.error('Erreur lors de l\'ajout du contact.', {
          position: 'top-right',
          autoClose: 3000,
          hideProgressBar: false,
          closeOnClick: true,
          pauseOnHover: true,
          draggable: true,
          progress: undefined,
        });
      }
    } catch (error) {
      toast.error('Erreur lors de la requête d\'ajout de contact.', {
        position: 'top-right',
        autoClose: 3000,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
        progress: undefined,
      });
    }

    // Fermez le modal après l'ajout du contact, si nécessaire
    setShowAddContactModal(false);
  };

  return (
    <>
      <div className="App">
        <Navbar bg="primary" data-bs-theme="dark">
          <Container>
            <FontAwesomeIcon icon={faAddressBook} size="2x" color="white"
/>    
            <Navbar.Brand href="#home">Contacts Manager</Navbar.Brand>
```

```
            <Nav className="me-auto">
              <Nav.Link href="#home">V1.0</Nav.Link>
            </Nav>
          </Container>
        </Navbar>

        <div className='container'>
          <div className='p-4'></div>
          <Button variant="danger" onClick={() => setShowAddContactModal(true)}>
            Ajouter un contact
          </Button>

          <AddContactModal
            show={showAddContactModal}
            handleClose={() => setShowAddContactModal(false)}
            handleAddContact={handleAddContact}
          />
          <ContactsTable contacts={contactsData} />
        </div>
      </div>
      <ToastContainer />
    </>
  );
}

export default App;
```

Modifier le code de App.js pour mettre a jour le compteur des contact

```
import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import React, { useState, useEffect } from 'react';
import AddContactModal from './components/AddContactModal';
import ContactsTable from './components/ContactsTable';
import { Navbar, Nav, Container, Button } from 'react-bootstrap';
import '@fortawesome/fontawesome-svg-core/styles.css';
import { library } from '@fortawesome/fontawesome-svg-core';
import { faAddressBook } from '@fortawesome/free-solid-svg-icons';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';

library.add(faAddressBook);

function App() {
  const [contactsData, setContactsData] = useState([]);

  const updateContactCount = (newContactCount) => {
    setContactsData((prevContacts) => {
      // Assuming contactsData is an array, update the contact count
      return prevContacts.map((contact) => ({ ...contact }));
    });
  };

  const [showAddContactModal, setShowAddContactModal] = useState(false);

  const fetchContacts = async () => {
    try {
      const response = await fetch('http://localhost:3001/contact/lister');
      if (!response.ok) {
        throw new Error(`Erreur lors de la récupération des contacts:
${response.statusText}`);
      }
      const data = await response.json();
      // Suppose que la liste des contacts est dans un objet nommé "liste"
      setContactsData(data.liste || []);
    } catch (error) {
      console.error(error.message);
    }
  };
```

```
  useEffect(() => {
    fetchContacts();
  }, []);

  const handleAddContact = async (newContact) => {
    try {
      const response = await fetch('http://localhost:3001/contact/ajouter', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(newContact),
      });

      if (response.ok) {
        // La requête a réussi, mettez à jour votre liste de contacts
        fetchContacts(); // Actualisez la liste des contacts après l'ajout
        toast.success('Contact ajouté avec succès.', {
          position: 'top-right',
          autoClose: 3000, // Duration in milliseconds
          hideProgressBar: false,
          closeOnClick: true,
          pauseOnHover: true,
          draggable: true,
          progress: undefined,
        });
      } else {
        // La requête a échoué, traitez les erreurs si nécessaire
        toast.error('Erreur lors de l\'ajout du contact.', {
          position: 'top-right',
          autoClose: 3000,
          hideProgressBar: false,
          closeOnClick: true,
          pauseOnHover: true,
          draggable: true,
          progress: undefined,
        });
      }
    } catch (error) {
      toast.error('Erreur lors de la requête d\'ajout de contact.', {
        position: 'top-right',
        autoClose: 3000,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
        progress: undefined,
      });
    }

    // Fermez le modal après l'ajout du contact, si nécessaire
    setShowAddContactModal(false);
  };

  return (
    <>
      <div className="App">
        <Navbar bg="primary" data-bs-theme="dark">
          <Container>
            <FontAwesomeIcon icon={faAddressBook} size="2x" color="white"
/>    
            <Navbar.Brand href="#home">Contacts Manager</Navbar.Brand>
            <Nav className="me-auto">
              <Nav.Link href="#home">V1.0</Nav.Link>
            </Nav>
          </Container>
        </Navbar>

        <div className='container'>
          <div className='p-4'></div>
          <Button variant="danger" onClick={() => setShowAddContactModal(true)}>
            Ajouter un contact
```

```
          </Button>

          <AddContactModal
            show={showAddContactModal}
            handleClose={() => setShowAddContactModal(false)}
            handleAddContact={handleAddContact}
            updateContactCount={updateContactCount}
          />
          <ContactsTable contacts={contactsData} />
        </div>
      </div>
      <ToastContainer />
    </>
  );
}

export default App;
```

Améliorer le formulaire ajouter contact pour ajouter un ensemble des tags

-Cote server (API)

Modifier dans **contactControllers** la méthode **ajouterContact**

```
exports.ajouterContact = async (req, res) => {
    try {
        // Assurez-vous que les données nécessaires sont présentes dans le corps de la requête
        const { nom, tel, tags } = req.body;
        if (!nom || !tel) {
            return res.status(400).json({
                message: 'Le nom et le téléphone sont requis.'
            });
        }

        // Créer un nouvel objet contact à partir des données de la requête
        const contact = new ContactModel({ nom, tel, tags });

        // Sauvegarder le contact dans la base de données
        const savedContact = await contact.save();

        // Retourner une réponse avec le contact ajouté
        return res.status(200).json({
            message: 'Contact ajouté avec succès.',
            contact: savedContact
        });
    } catch (err) {
        // Gérer les erreurs de manière appropriée
        console.error(err);
        res.status(500).json({ message: 'Erreur interne du serveur.' });
    }
};
```

-Cote FrondEnd (ReactJS)

Modifier dans **src/components/addContactModal.js**

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

const AddContactModal = ({ show, handleClose, handleAddContact }) => {
  const [nom, setNom] = useState('');
  const [tel, setTel] = useState('');
  const [tags, setTags] = useState(''); // Ajout de l'état pour les tags
```

```
  const handleSave = () => {
    // Validez les données si nécessaire

    // Appel de la fonction pour ajouter le contact avec les tags
    handleAddContact({ nom, tel, tags: tags.split(',').map(tag => tag.trim()) });

    // Réinitialisez les champs du formulaire
    setNom('');
    setTel('');
    setTags('');

    // Fermez le modal
    handleClose();
  };

  return (
    <Modal show={show} onHide={handleClose}>
      <Modal.Header closeButton>
        <Modal.Title>Ajouter un contact</Modal.Title>
      </Modal.Header>
      <Modal.Body>
        <Form>
          <Form.Group controlId="nom">
            <Form.Label>Nom</Form.Label>
            <Form.Control
              type="text"
              placeholder="Entrez le nom"
              value={nom}
              onChange={(e) => setNom(e.target.value)}
            />
          </Form.Group>
          <Form.Group controlId="tel">
            <Form.Label>Téléphone</Form.Label>
            <Form.Control
              type="text"
              placeholder="Entrez le numéro de téléphone"
              value={tel}
              onChange={(e) => setTel(e.target.value)}
            />
          </Form.Group>
          <Form.Group controlId="tags">
            <Form.Label>Tags (séparés par des virgules)</Form.Label>
            <Form.Control
              type="text"
              placeholder="Entrez les tags"
              value={tags}
              onChange={(e) => setTags(e.target.value)}
            />
          </Form.Group>
        </Form>
      </Modal.Body>
      <Modal.Footer>
        <Button variant="secondary" onClick={handleClose}>
          Annuler
        </Button>
        <Button variant="primary" onClick={handleSave}>
          Enregistrer
        </Button>
      </Modal.Footer>
    </Modal>
  );
};

export default AddContactModal;
```

Apercu

```
{
  "_id": {…},
  "nom": "kamel abbassi",
  "tel": "26388202",
  "tags": [
    "ami",
    "travail",
    "job"
  ],
  "createdAt": {…},
  "updatedAt": {…},
  "__v": 0
}
```


Ajouter le bouton Supprimer dans le fichier ContactTable.js

```
//ContactsTable.js
import React, { useState } from 'react';
import { Table, Pagination, Badge, Form, Button } from 'react-bootstrap';
import * as XLSX from 'xlsx';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faFileExcel , faFilePdf} from '@fortawesome/free-solid-svg-icons';
import DeleteContactModal from './DeleteContactModal';
import axios from 'axios';
// JsPDF
import jsPDF from 'jspdf';
import 'jspdf-autotable';

const ContactsTable = ({ contacts }) => {
  const [currentPage, setCurrentPage] = useState(1);
  const [sortBy, setSortBy] = useState(null);
  const [sortOrder, setSortOrder] = useState('asc');
  const [searchTerm, setSearchTerm] = useState('');
  const contactsPerPage = 5;

  const [showDeleteModal, setShowDeleteModal] = useState(false);
  const [contactIdToDelete, setContactIdToDelete] = useState(null);

  if (!contacts || contacts.length === 0) {
    return <p>No contacts available.</p>;
  }

  const indexOfLastContact = currentPage * contactsPerPage;
  const indexOfFirstContact = indexOfLastContact - contactsPerPage;
  const sortedContacts = contacts.filter(
    (contact) =>
      contact.nom.toLowerCase().includes(searchTerm.toLowerCase()) ||
      contact.tel.toString().includes(searchTerm)
  ).sort((a, b) => {
    if (sortBy === 'numero') {
      return sortOrder === 'asc' ? a.numero - b.numero : b.numero - a.numero;
    } else if (sortBy === 'nom') {
      return sortOrder === 'asc' ? a.nom.localeCompare(b.nom) :
b.nom.localeCompare(a.nom);
    } else if (sortBy === 'tel') {
      return sortOrder === 'asc' ? a.tel - b.tel : b.tel - a.tel;
    }
    // Ajoutez d'autres critères de tri si nécessaire
    return 0;
  });

  const currentContacts = sortedContacts.slice(indexOfFirstContact,
indexOfLastContact);
  const totalPages = Math.ceil(sortedContacts.length / contactsPerPage);

  const paginate = (pageNumber) => setCurrentPage(pageNumber);
  const handleSort = (key) => {
```

```
    if (sortBy === key) {
      setSortOrder(sortOrder === 'asc' ? 'desc' : 'asc');
    } else {
      setSortBy(key);
      setSortOrder('asc');
    }
  };

  //Mettre en forme le terme cherché
  const highlightText = (text, query) => {
    if (typeof text !== 'string') {
      text = String(text);
    }
    const parts = text.split(new RegExp(`(${query})`, 'gi'));
    return parts.map((part, i) =>
      part.toLowerCase() === query.toLowerCase() ? <strong key={i}>{part}</strong>
: part
    );
  };

  const handleExportExcel = () => {
    const dataToExport = sortedContacts.map((contact) => ({
      Numéro: indexOfFirstContact + sortedContacts.indexOf(contact) + 1,
      Avatar: contact.avatarUrl || 'https://picsum.photos/200/300',
      Nom: contact.nom,
      Téléphone: contact.tel,
    }));

    const ws = XLSX.utils.json_to_sheet(dataToExport);
    const wb = XLSX.utils.book_new();
    XLSX.utils.book_append_sheet(wb, ws, 'Contacts');
    XLSX.writeFile(wb, 'contacts.xlsx');
  };

// Function to handle exporting to PDF
const handleExportPDF = () => {
  // Create a new jsPDF instance
  const doc = new jsPDF();

  // Set text color to red
  doc.setTextColor(255, 0, 0);

  // Set font style to bold
  doc.setFont('bold');

  // Add title in red and bold
  doc.setFontSize(20);
  doc.text('Liste des contacts', 100, 10, { align: 'center' });

  // Reset text color and font style
  doc.setTextColor(0, 0, 0);
  doc.setFont('normal');

  // Add date and time at the top of the page
  const today = new Date();
  const newdat = "Imprimé le: " + today.toLocaleString();
  doc.text(newdat, 10, 20);

  // Add an image at a specific location on the page
  var img = new Image();
  img.src = 'img/elife.jpg';
  doc.addImage(img, 'png', 150, 10, 20, 15);

  // Add some space before the table
  const spaceBeforeTable = 30;

  doc.autoTable({
    head: [['Numéro', 'Avatar', 'Nom', 'Téléphone']],
    body: contacts.map((contact, index) => [
      indexOfFirstContact + index + 1,
      { image: contact.avatarUrl || 'https://picsum.photos/200/300', width: 20,
height: 20 },
      contact.nom,
```

```
      contact.tel,
    ]),
    startY: spaceBeforeTable, // Adjust the Y-coordinate to add space

    margin: { top: 10 }, // Add extra margin to accommodate the header
  });

  // Save the PDF with the specified filename
  doc.save('contacts.pdf');
};

const handleDeleteClick = (contactId) => {
  setContactIdToDelete(contactId);
  setShowDeleteModal(true);
};

const handleDeleteContact = async (contactId) => {
  try {
    // Make the API call to delete the contact
    await axios.delete(`http://localhost:3001/contact/${contactId}/supprimer`);

    // Handle success, e.g., update your contact list
    console.log('Contact deleted successfully');
    window.location.reload(); // Reload the entire page
  } catch (error) {
    // Handle error, show an error message, etc.
    console.error('Error deleting contact', error);
  }
};

const handleCloseDeleteModal = () => {
  setShowDeleteModal(false);
  setContactIdToDelete(null);
};



  return (
    <>
      <h2 className="text-muted">
        Votre liste des contacts contient{' '}
        <Badge variant="primary">{contacts.length}</Badge> contacts
      </h2>

      <p>
        contacts triés par: <span class="badge bg-primary">{sortBy ? "" + sortBy :
""}</span>
      </p>

      <Form>
        <Form.Group controlId="searchTerm">
          <Form.Control
            type="text"
            placeholder="Rechercher par nom"
            value={searchTerm}
            onChange={(e) => setSearchTerm(e.target.value)}
          />
        </Form.Group>
      </Form>
      <div className='m-2'></div>
      <Button variant="success" onClick={handleExportExcel}>
        <FontAwesomeIcon icon={faFileExcel} className="mr-1" /> Exporter en Excel
      </Button>
       

      <Button variant="primary" onClick={handleExportPDF} className="ml-2">
        <FontAwesomeIcon icon={faFilePdf} className="mr-1" /> Exporter en PDF
      </Button>
      <div className='m-2'></div>
      <Table striped bordered hover>
        <thead>
          <tr>
            <th onClick={() => handleSort('numero')}>Numéro</th>
```

```
            <th>Avatar</th>
            <th onClick={() => handleSort('nom')}>Nom</th>
            <th onClick={() => handleSort('tel')}>Téléphone</th>
          </tr>
        </thead>
        <tbody>
          {currentContacts.map((contact, index) => (
            <tr key={index}>
              <td>{indexOfFirstContact + index + 1}</td>
              <td>
                <img
                  src={contact.avatarUrl || 'https://picsum.photos/200/300'}
                  alt={`Avatar de ${contact.nom}`}
                  width="50"
                  height="50"
                />
              </td>
              <td>
                {highlightText(contact.nom, searchTerm)}
              </td>
              <td>
                {highlightText(contact.tel, searchTerm)}
              </td>
              <td>
                {/* Example button to trigger the modal */}
                <Button variant="danger" onClick={() =>
handleDeleteClick(contact._id)}>
                  Supprimer le contact
                </Button>

                {/* DeleteContactModal usage */}
                {showDeleteModal && (
                  <DeleteContactModal
                    show={showDeleteModal}
                    handleClose={handleCloseDeleteModal}
                    deleteContact={handleDeleteContact}
                    contactId={contactIdToDelete}

                  />
                )}
              </td>


            </tr>
          ))}
        </tbody>
      </Table>

      <Pagination>
        {Array.from({ length: totalPages }).map((_, index) => (
          <Pagination.Item
            key={index + 1}
            active={index + 1 === currentPage}
            onClick={() => paginate(index + 1)}
          >
            {index + 1}
          </Pagination.Item>
        ))}
      </Pagination>
    </>
  );
};

export default ContactsTable;
```

Ajouter le composant **DeleteContactModal**

```
//DeleteContactModal
import React from 'react';
import { Modal, Button } from 'react-bootstrap';
import { toast } from 'react-toastify';

const DeleteContactModal = ({ show, handleClose, deleteContact, contactId }) => {
    const handleDelete = async () => {

        try {
            // Appeler la fonction pour supprimer le contact
            await deleteContact(contactId); // Pass the contactId to the
deleteContact function

            toast.success('Contact supprimé avec succès.', {
                position: 'top-right',
                autoClose: 3000,
                hideProgressBar: false,
                closeOnClick: true,
                pauseOnHover: true,
                draggable: true,
                progress: undefined,
            });
            handleClose();
        } catch (error) {
            toast.error('Erreur lors de la suppression du contact.', {
                position: 'top-right',
                autoClose: 3000,
                hideProgressBar: false,
                closeOnClick: true,
                pauseOnHover: true,
                draggable: true,
                progress: undefined,
            });
        }
    };

    return (
        <Modal show={show} onHide={handleClose}>
            <Modal.Header closeButton>
                <Modal.Title>Supprimer le contact</Modal.Title>
            </Modal.Header>
            <Modal.Body>
                <p>Êtes-vous sûr de vouloir supprimer ce contact?</p>
            </Modal.Body>
            <Modal.Footer>
                <Button variant="secondary" onClick={handleClose}>
                    Annuler
                </Button>
                <Button variant="danger" onClick={handleDelete}>
                    Supprimer
                </Button>
            </Modal.Footer>
        </Modal>
    );
};

export default DeleteContactModal;
```

Ajouter composant pour la **modification**

```
// EditContactModal.js
import React, { useState } from 'react';
import { Modal, Button, Form } from 'react-bootstrap';
import { toast } from 'react-toastify';

const EditContactModal = ({ show, handleClose, editContact, contact }) => {
  const [editedData, setEditedData] = useState({
    nom: contact.nom,
    tel: contact.tel,
    // Add other fields as needed
  });
```

```
  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setEditedData((prevData) => ({
      ...prevData,
      [name]: value,
    }));
  };

  const handleEdit = async () => {
    try {
      // Perform validation or additional logic as needed

      // Call the function to edit the contact
      await editContact(contact._id, editedData);

      toast.success('Contact modifié avec succès.', {
        position: 'top-right',
        autoClose: 3000,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
        progress: undefined,
      });
      handleClose();
    } catch (error) {
      toast.error('Erreur lors de la modification du contact.', {
        position: 'top-right',
        autoClose: 3000,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
        progress: undefined,
      });
    }
  };

  return (
    <Modal show={show} onHide={handleClose}>
      <Modal.Header closeButton>
        <Modal.Title>Modifier le contact</Modal.Title>
      </Modal.Header>
      <Modal.Body>
        <Form>
          <Form.Group controlId="editNom">
            <Form.Label>Nom</Form.Label>
            <Form.Control
              type="text"
              placeholder="Entrez le nouveau nom"
              name="nom"
              value={editedData.nom}
              onChange={handleInputChange}
            />
          </Form.Group>

          <Form.Group controlId="editTel">
            <Form.Label>Téléphone</Form.Label>
            <Form.Control
              type="tel"
              placeholder="Entrez le nouveau numéro de téléphone"
              name="tel"
              value={editedData.tel}
              onChange={handleInputChange}
            />
          </Form.Group>

          {/* Add other form fields for editing other contact details */}

          <Button variant="primary" onClick={handleEdit}>
            Enregistrer les modifications
          </Button>
        </Form>
```

```
      </Modal.Body>
      <Modal.Footer>
        <Button variant="secondary" onClick={handleClose}>
          Annuler
        </Button>
      </Modal.Footer>
    </Modal>
  );
};

export default EditContactModal;
```

Modifier ContactTable.js

```
// ContactsTable.js
import React, { useState, useRef } from 'react';
import { Table, Pagination, Badge, Form, Button } from 'react-bootstrap';
import * as XLSX from 'xlsx';
import { useReactToPrint } from 'react-to-print';
import DeleteContactModal from './DeleteContactModal';
import EditContactModal from './EditContactModal';
import axios from 'axios';

import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faTrash, faEdit } from '@fortawesome/free-solid-svg-icons';

const ContactsTable = ({ contacts, fetchContacts }) => {
  const [currentPage, setCurrentPage] = useState(1);
  const [sortBy, setSortBy] = useState(null);
  const [sortOrder, setSortOrder] = useState('asc');
  const [searchTerm, setSearchTerm] = useState('');
  const [showDeleteModal, setShowDeleteModal] = useState(false);
  const [contactIdToDelete, setContactIdToDelete] = useState(null);

  const [showEditModal, setShowEditModal] = useState(false);
  const [contactToEdit, setContactToEdit] = useState(null);

  const contactsPerPage = 5;
  const componentRef = useRef();

  const handlePrint = useReactToPrint({
    content: () => componentRef.current,
  });

  if (!contacts || contacts.length === 0) {
    return <p>No contacts available.</p>;
  }

  const indexOfLastContact = currentPage * contactsPerPage;
  const indexOfFirstContact = indexOfLastContact - contactsPerPage;
  const sortedContacts = contacts
    .filter((contact) => {
      const fullName = `${contact.nom}`.toLowerCase();
      return fullName.includes(searchTerm.toLowerCase());
    })
    .sort((a, b) => {
      if (sortBy === 'numero') {
        return sortOrder === 'asc' ? a.numero - b.numero : b.numero - a.numero;
      } else if (sortBy === 'nom') {
        return sortOrder === 'asc' ? a.nom.localeCompare(b.nom) :
b.nom.localeCompare(a.nom);
      } else if (sortBy === 'tel') {
        return sortOrder === 'asc' ? a.tel - b.tel : b.tel - a.tel;
      }
      // Add other sorting criteria if necessary
      return 0;
    });

  const currentContacts = sortedContacts.slice(indexOfFirstContact,
indexOfLastContact);
  const totalPages = Math.ceil(sortedContacts.length / contactsPerPage);
```

```
  const paginate = (pageNumber) => setCurrentPage(pageNumber);
  const handleSort = (key) => {
    if (sortBy === key) {
      setSortOrder(sortOrder === 'asc' ? 'desc' : 'asc');
    } else {
      setSortBy(key);
      setSortOrder('asc');
    }
  };

  const handleExportExcel = () => {
    const dataToExport = sortedContacts.map((contact) => ({
      Numéro: indexOfFirstContact + sortedContacts.indexOf(contact) + 1,
      Avatar: contact.avatarUrl || 'https://picsum.photos/200/300',
      Nom: contact.nom,
      Téléphone: contact.tel,
    }));

    const ws = XLSX.utils.json_to_sheet(dataToExport);
    const wb = XLSX.utils.book_new();
    XLSX.utils.book_append_sheet(wb, ws, 'Contacts');
    XLSX.writeFile(wb, 'contacts.xlsx');
  };

  const handleDeleteClick = (contactId) => {
    setContactIdToDelete(contactId);
    setShowDeleteModal(true);
  };

  const handleDeleteContact = async (contactId) => {
    try {
      await axios.delete(`http://localhost:3001/contact/${contactId}/supprimer`);
      fetchContacts();
      console.log('Contact deleted successfully');
    } catch (error) {
      console.error('Error deleting contact', error);
    } finally {
      setShowDeleteModal(false);
      setContactIdToDelete(null);
    }
  };

  const handleCloseDeleteModal = () => {
    setShowDeleteModal(false);
    setContactIdToDelete(null);
  };

  const handleEditClick = (contact) => {
    setContactToEdit(contact);
    setShowEditModal(true);
  };

  const handleEditContact = async (contactId, editedData) => {
    try {
      await axios.put(`http://localhost:3001/contact/${contactId}/modifier`,
editedData);
      fetchContacts();
      console.log('Contact edited successfully');
    } catch (error) {
      console.error('Error editing contact', error);
    } finally {
      setShowEditModal(false);
      setContactToEdit(null);
    }
  };

  const handleCloseEditModal = () => {
    setShowEditModal(false);
    setContactToEdit(null);
  };

  return (
    <>
```

```
      <h2 className="text-muted">
        Votre liste des contacts contient{' '}
        <Badge variant="primary">{contacts.length}</Badge> contacts
      </h2>

      <Form>
        <Form.Group controlId="searchTerm">
          <Form.Control
            type="text"
            placeholder="Rechercher par nom"
            value={searchTerm}
            onChange={(e) => setSearchTerm(e.target.value)}
          />
        </Form.Group>
      </Form>

      <Button variant="success" onClick={handleExportExcel}>
        Exporter en Excel
      </Button>  
      <Button variant="primary" onClick={handlePrint}>
        Imprimer en PDF
      </Button>
      <div className='m-2'></div>

      <Table striped bordered hover ref={componentRef}>
        <thead>
          <tr>
            <th onClick={() => handleSort('numero')}>Numéro</th>
            <th>Avatar</th>
            <th onClick={() => handleSort('nom')}>Nom</th>
            <th onClick={() => handleSort('tel')}>Téléphone</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          {currentContacts.map((contact, index) => (
            <tr key={index}>
              <td>{indexOfFirstContact + index + 1}</td>
              <td>
                <img
                  src={contact.avatarUrl || 'img/inconnu.jpg'}
                  alt={`Avatar de ${contact.nom}`}
                  width="50"
                  height="50"
                />
              </td>
              <td>{contact.nom}</td>
              <td>{contact.tel}</td>
              <td>
                <Button variant="danger" onClick={() =>
handleDeleteClick(contact._id)} className="mr-2">
                  <FontAwesomeIcon icon={faTrash} className="mr-1" /> Supprimer le
contact
                </Button>
                <Button variant="info" onClick={() => handleEditClick(contact)}
className="ml-2">
                  <FontAwesomeIcon icon={faEdit} className="mr-1" /> Modifier le
contact
                </Button>
              </td>

            </tr>
          ))}
        </tbody>
      </Table>

      <Pagination>
        {Array.from({ length: totalPages }).map((_, index) => (
          <Pagination.Item
            key={index + 1}
            active={index + 1 === currentPage}
            onClick={() => paginate(index + 1)}
          >
```

```
            {index + 1}
          </Pagination.Item>
        ))}
      </Pagination>

      {/* DeleteContactModal usage */}
      {showDeleteModal && (
        <DeleteContactModal
          show={showDeleteModal}
          handleClose={handleCloseDeleteModal}
          deleteContact={handleDeleteContact}
          contactId={contactIdToDelete}
        />
      )}

      {/* EditContactModal usage */}
      {showEditModal && (
        <EditContactModal
          show={showEditModal}
          handleClose={handleCloseEditModal}
          editContact={handleEditContact}
          contact={contactToEdit}
        />
      )}
    </>
  );
};

export default ContactsTable;
```

Modifier le fichier App.js

```
import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import React, { useState, useEffect } from 'react';
import AddContactModal from './components/AddContactModal';
import ContactsTable from './components/ContactsTable';
import { Navbar, Nav, Container, Button } from 'react-bootstrap';
import '@fortawesome/fontawesome-svg-core/styles.css';
import { library } from '@fortawesome/fontawesome-svg-core';
import { faAddressBook } from '@fortawesome/free-solid-svg-icons';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { ToastContainer} from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';

library.add(faAddressBook);

function App() {
  const [contactsData, setContactsData] = useState([]);
  const [showAddContactModal, setShowAddContactModal] = useState(false);

  const fetchContacts = async () => {
    try {
      const response = await fetch('http://localhost:3001/contact/lister');
      if (!response.ok) {
        throw new Error(`Erreur lors de la récupération des contacts:
${response.statusText}`);
      }
      const data = await response.json();
      setContactsData(data.liste || []);
    } catch (error) {
      console.error(error.message);
    }
  };

  useEffect(() => {
    fetchContacts();
  }, []);

  return (
    <>
      <div className="App">
```

```
        <Navbar bg="primary" data-bs-theme="dark">
          <Container>
            <FontAwesomeIcon icon={faAddressBook} size="2x" color="white"
/>    
            <Navbar.Brand href="#home">Contacts Manager</Navbar.Brand>
            <Nav className="me-auto">
              <Nav.Link href="#home">V1.0</Nav.Link>
            </Nav>
          </Container>
        </Navbar>

        <div className='container'>
          <div className='p-4'></div>
          <Button variant="danger" onClick={() => setShowAddContactModal(true)}>
            Ajouter un contact
          </Button>

          <AddContactModal
            show={showAddContactModal}
            handleClose={() => setShowAddContactModal(false)}
            fetchContacts={fetchContacts} // Passer la fonction fetchContacts au
composant enfant
          />
          <ContactsTable contacts={contactsData} fetchContacts={fetchContacts} />
        </div>
      </div>
      <ToastContainer />
    </>
  );
}

export default App;
```

Temps réel

Pour mettre en place des mises à jour en temps réel pour votre liste de tâches lors de modifications via l'API, vous pouvez envisager d'utiliser une solution de gestion d'état telle que Redux ou le React Context API. Voici une vue d'ensemble de la manière dont vous pouvez aborder cela :

1. **Configurer la gestion d'état :**
    - Installez une bibliothèque de gestion d'état comme Redux ou utilisez le React Context API.
2. **Créer des actions :**
    - Définissez des actions pour récupérer, supprimer et mettre à jour les contacts.
3. **Créer des réducteurs :**
    - Créez des réducteurs pour gérer les modifications d'état en fonction des actions.
4. **Déclencher des actions :**
    - Déclenchez des actions dans votre composant pour provoquer des modifications d'état.
5. **Mettre à jour le composant :**
    - Mettez à jour votre composant pour utiliser l'état fourni par la solution de gestion d'état.

Le code complet avec Redux pourrait ressembler à ceci. Assurez-vous d'installer les dépendances nécessaires en exécutant npm install redux react-redux axios si elles ne sont pas déjà installées.

Installer redux react-redux

```
npm install redux react-redux
```

Créer les actions (actions.js) :

```
// actions.js
export const FETCH_CONTACTS_SUCCESS = 'FETCH_CONTACTS_SUCCESS';

export const fetchContactsSuccess = (contacts) => ({
  type: FETCH_CONTACTS_SUCCESS,
  payload: contacts,
});
```

Créer les réducteurs (reducers.js) :

javascript

Copy code

// reducers.js

import { FETCH_CONTACTS_SUCCESS } from './actions';

const initialState = {

contacts: [],

};

const contactsReducer = (state = initialState, action) => {

switch (action.type) {

case FETCH_CONTACTS_SUCCESS:

return {

...state,

contacts: action.payload,

};

// Gérer d'autres actions si nécessaire

default:

return state;

}

};

export default contactsReducer;

Créer le magasin Redux (store.js) :

javascript

Copy code

```
// store.js
import { createStore } from 'redux';
import contactsReducer from './reducers';

const store = createStore(contactsReducer);

export default store;
```

Modifier le composant (ContactsTable.js) pour utiliser Redux :

javascript

Copy code

```
// ContactsTable.js
import React, { useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { fetchContactsSuccess } from './actions';
import axios from 'axios';

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

  const fetchContacts = async () => {
    try {
      const response = await axios.get('http://localhost:3001/contacts');
      dispatch(fetchContactsSuccess(response.data));
    } catch (error) {
      console.error('Erreur lors de la récupération des contacts', error);
    }
  };
```

```
  useEffect(() => {
    fetchContacts();
  }, []); // Récupérer les contacts lors du montage du composant

  // ... (autres parties du code restent inchangées)

  return (
    // ... (votre code JSX reste inchangé)
  );
};

export default ContactsTable;
```

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.

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