src/dataService.js

```
// Exporting an asynchronous function to fetch data from a JSON
file
export const fetchData = async () => {
    // Fetch the JSON file located in the public directory of the
React app
    const response = await fetch('/data.json');

    // Convert the response to JSON format
    const data = await response.json();

    // Return the parsed data to be used in the React component
    return data;
};
```

Explanation:

1. export const fetchData = async () => {...}:
    - **Exporting**: This function is exported so it can be used in other files (App.js in this case).
    - **Async Function**: Declaring the function as async allows the use of await inside it, enabling asynchronous operations.
2. const response = await fetch('/data.json');:
    - **fetch('/data.json')**: The fetch function is a built-in browser API used to make network requests. Here, it fetches the data.json file from the **public** directory.
    - **await**: Pauses the function execution until the fetch is completed. This ensures response is assigned only after the data is fully loaded.
3. const data = await response.json();:
    - **response.json()**: This method converts the fetched data (which is in raw format) into a **JSON object**.
    - Again, await is used to handle this asynchronous conversion.
4. return data;:
    - The function returns the **parsed JSON data**, which is typically an object or an array, to wherever this function is called.

```javascript
export const saveData = (data) => {
    // Create a temporary anchor (link) element
    const a = document.createElement('a');

    // Convert the data object to a JSON string and create a downloadable file
    const file = new Blob([JSON.stringify(data, null, 2)], { type: 'application/json' });

    // Generate a URL for the file and set it as the href of the anchor element
    a.href = URL.createObjectURL(file);

    // Set the default file name for the download
    a.download = 'data.json';

    // Programmatically trigger a click on the anchor element to start the download
    a.click();
};
```

Explanation:

1. export const saveData = (data) => {...}:
   - **Exporting**: Like fetchData, this function is also exported for use in other components.
   - **Parameter data**: The data parameter is expected to be a **JavaScript object** (e.g., the updated user list).
2. const a = document.createElement('a');:
   - Creates a **temporary HTML anchor (<a>) element**.
   - This anchor element is not added to the DOM visually but is used to trigger a file download.
3. const file = new Blob([JSON.stringify(data, null, 2)], { type: 'application/json' });:
   - **JSON.stringify(data, null, 2)**:
     - Converts the data object into a **JSON-formatted string**.
     - null and 2 are formatting parameters:
       - null: Replacer function (not used here).
       - 2: Adds **indentation** of 2 spaces to make the JSON more readable.
   - **new Blob([...], { type: 'application/json' })**:
     - Creates a **Blob (Binary Large Object)**, which acts like a file in the browser.

        - The type is set to 'application/json', indicating the file format.

4. a.href = URL.createObjectURL(file);:
    - **URL.createObjectURL(file)**:
        - Generates a **temporary URL** pointing to the file (Blob) created.
    - **a.href**:
        - Sets this URL as the href attribute of the anchor element.
        - When the link is clicked, it triggers a **download** instead of opening the file.
5. a.download = 'data.json';:
    - Sets the **default filename** for the download as data.json.
    - When the download starts, the file will be saved with this name.
6. a.click();:
    - Programmatically **triggers a click** event on the anchor element.
    - This **automatically starts the download** without requiring user interaction.

Concepts:

- **Blob**: Represents immutable raw data. Used to handle files in the browser.
- **URL.createObjectURL()**: Creates a temporary URL that points to a Blob or File object.
- **Programmatic Download**: The a.click() technique is often used to download files from frontend code without needing a server.

Practical Use Case:

- When you modify data in your React app (e.g., adding a new user to a list), saveData will download the updated data.json file to your machine.
- Useful for **exporting data** from a web app, like downloading user data, configurations, or reports.

src/App.js

```
// Importing necessary modules from React
import React, { useEffect, useState } from 'react';

// Importing custom data fetching and saving functions from
dataService.js
import { fetchData, saveData } from './dataService';
```

Explanation:

1. import React, { useEffect, useState } from 'react';:
   - **React**: The core library for building UI components.
   - **useEffect**: A React Hook for performing **side effects** (e.g., data fetching) in functional components.
   - **useState**: A Hook for **state management** within a functional component.
2. import { fetchData, saveData } from './dataService';:
   - Imports fetchData to **load data** from data.json.
   - Imports saveData to **save modified data** back to the JSON file.

```
function App() {
    // Declaring a state variable 'users' to store user data with
an initial value of an empty array
    const [users, setUsers] = useState([]);
```

Explanation:

- **useState([])**:
  - Initializes users as an **empty array**.
  - setUsers is a **setter function** used to update the users state.
  - Any update to users will trigger a **re-render** of the component.

```
   useEffect(() => {
        // Parse (read) data from JSON
        fetchData().then(data => {
            setUsers(data.users);
        });
    }, []);
```

Explanation:

1. useEffect(() => {...}, []);:
   - Executes the provided function **once** when the component **mounts**.
   - The empty array ([]) as the **dependency array** means this effect runs **only once**.
2. fetchData().then(data => {...}):
   - Calls the **async fetchData function** from dataService.js.
   - **.then(data => {...})** handles the **promise** returned by fetchData.
3. setUsers(data.users);:
   - **Updates** the users state with the data fetched from the data.json file.
   - **Triggers a re-render**, displaying the fetched user list on the UI.

```
// Add new user and save to JSON
  const addNewUser = () => {
      const newUser = { name: 'Priya', age: 28, city: 'Delhi' };
      const updatedUsers = [...users, newUser];
      setUsers(updatedUsers);

      // Save updated data back to JSON
      const updatedData = { users: updatedUsers };
      saveData(updatedData);
  };
```

Explanation:

1. const addNewUser = () => {...}:
   - Defines a function to **add a new user** to the list and save the updated data.
2. const newUser = { name: 'Priya', age: 28, city: 'Delhi' };:
   - Creates a **new user object** with name, age, and city properties.
3. const updatedUsers = [...users, newUser];:
   - **Spread Operator (...users)**:
     - Creates a **new array** by copying all existing users.
     - Adds newUser to the **end of the array**.
   - **Why use spread operator?**
     - **Immutability**: Ensures users is not modified directly.
4. setUsers(updatedUsers);:
   - Updates the users state with the **new array**.
   - The **UI automatically updates** to reflect the changes.
5. const updatedData = { users: updatedUsers };:
   - Creates a **new object** in the same format as the data.json file.

    - The users array is nested under the users key to match the JSON structure.
6. saveData(updatedData);:
    - Calls saveData to **download** the updated data.json file with the **new user** included.

```
return (
        <div className="App">
            <h1>User List</h1>
            <ul>
                {users.map((user, index) => (
                    <li key={index}>{user.name} - {user.age} -
{user.city}</li>
                ))}
            </ul>
            <button onClick={addNewUser}>Add New User</button>
        </div>
    );
}
```

Explanation:

1. <div className="App">...</div>:
    - Main **container** for the component's content.
2. <h1>User List</h1>:
    - Displays a **heading** for the user list.
3. <ul>...</ul>:
    - Creates an **unordered list** for displaying users.
4. {users.map((user, index) => (...))}:
    - **.map() function**:
        - **Iterates** over the users array.
        - **Returns a list item** (<li>) for each user.
5. <li key={index}>...<li>:
    - **Displays user details** (name, age, city).
    - The **key prop** (index in this case) is necessary for **React to track** elements in a list.
6. <button onClick={addNewUser}>Add New User</button>:
    - Renders a **button**.
    - When clicked, **calls addNewUser** to add a new user and download the updated JSON.