

Day 11

React Lists and Keys



A Comprehensive Guide

What Are Lists in React?

A list in React refers to **a collection** of **similar data** that you want to render in your application.

It refers to a series of items that are dynamically generated from an array. Each item in the array is typically rendered as a React component, which makes lists a powerful tool for displaying repeating data.

For example, imagine you have an array of users, and you want to display their names as a list. React allows you to dynamically generate UI elements (like `` elements or cards) based on that array.

Rendering Lists in React

To render a list in React, we commonly use JavaScript's **.map()** method, which allows us to **loop** through an array and return a **new array** containing JSX elements.

Basic Example:

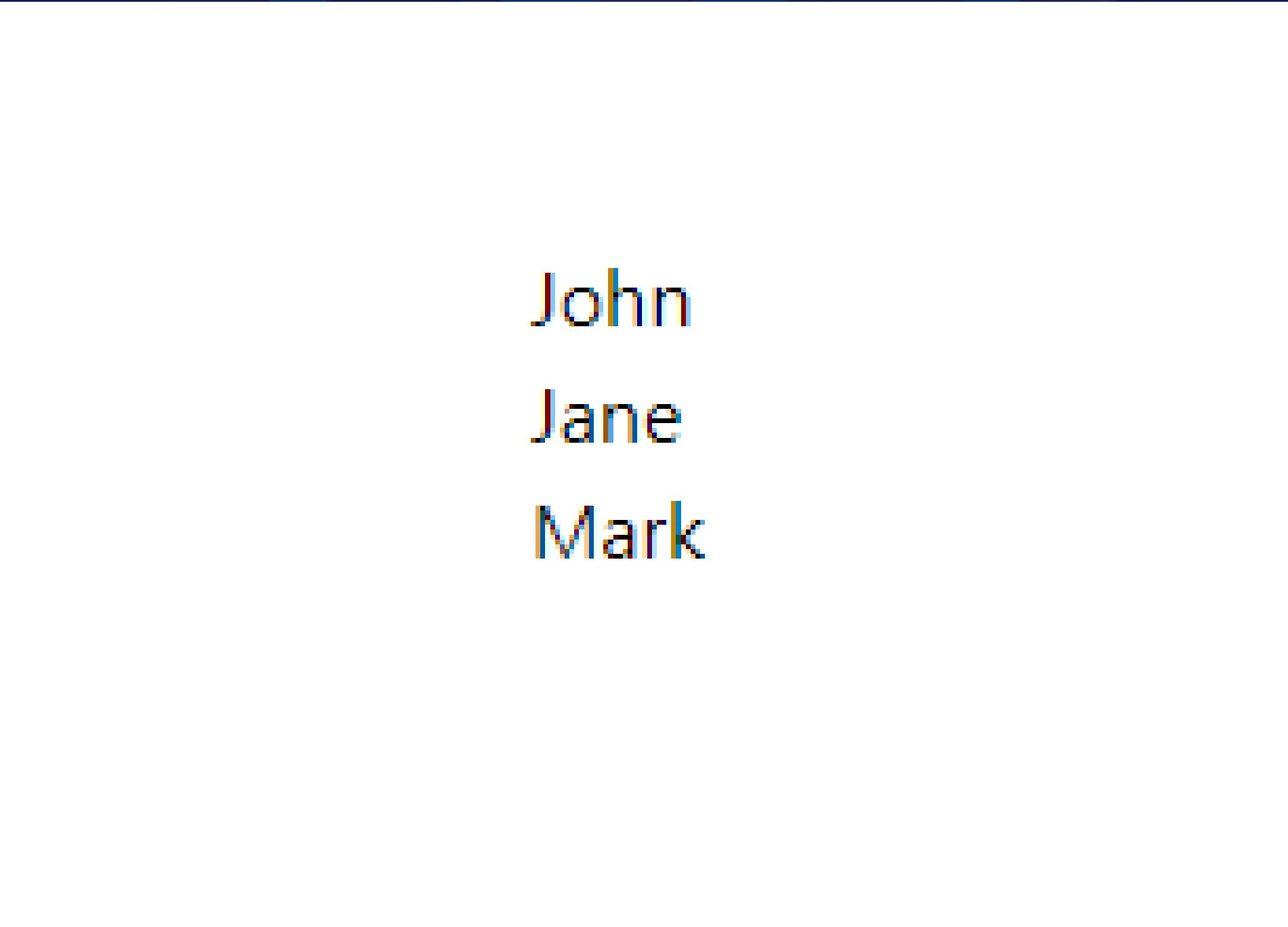


```
1 const users = ['John', 'Jane', 'Mark'];
2
3 function UserList() {
4   return (
5     <ul>
6       {users.map(user => <li key={user}>{user}</li>)}
7     </ul>
8   );
9 }
10
11 export default UserList;
```

the users array is accessed using .map()
users variable called

- We have an array of names (**users**), which we want to render inside a **** element.
- We use **map()** to iterate over the array and create an **** for each user.
- The **key** attribute (more on that later) is added to each **** to uniquely identify each item.

If you run this in your application, you should see the list displayed like this:



John

Jane

Mark

Keys in React

Keys are an important part of React when rendering lists.

They help React keep track of which items in a list have **changed**, **been added**, or **removed**, which leads to more efficient updates of the user interface (UI) and re-renders of components..

Without the key prop, React wouldn't know how to efficiently update the DOM, which could result in poor performance.

Always use a **unique value** for the key, such as an **id** from your data or **the index** of the item (though using **index** is generally **not recommended** if items can be reordered or deleted).

For Example:

```
● ● ●  
1 const users = [  
2   { id: 1, name: 'John' },  
3   { id: 2, name: 'Jane' },  
4   { id: 3, name: 'Mark' }  
5 ];  
6  
7 function UserList() {  
8   return (  
9     <ul>  
10       {users.map(user => (  
11         <li key={user.id}>{user.name}</li>  
12       ))}  
13     </ul>  
14   );  
15 }  
16  
17 export default UserList;
```

this will fetch in each user by their id

- We are using the **id** property of each user object as the **key**.
- Each user has a unique **id**, which ensures React can uniquely identify each list item.

Dynamic Lists Example

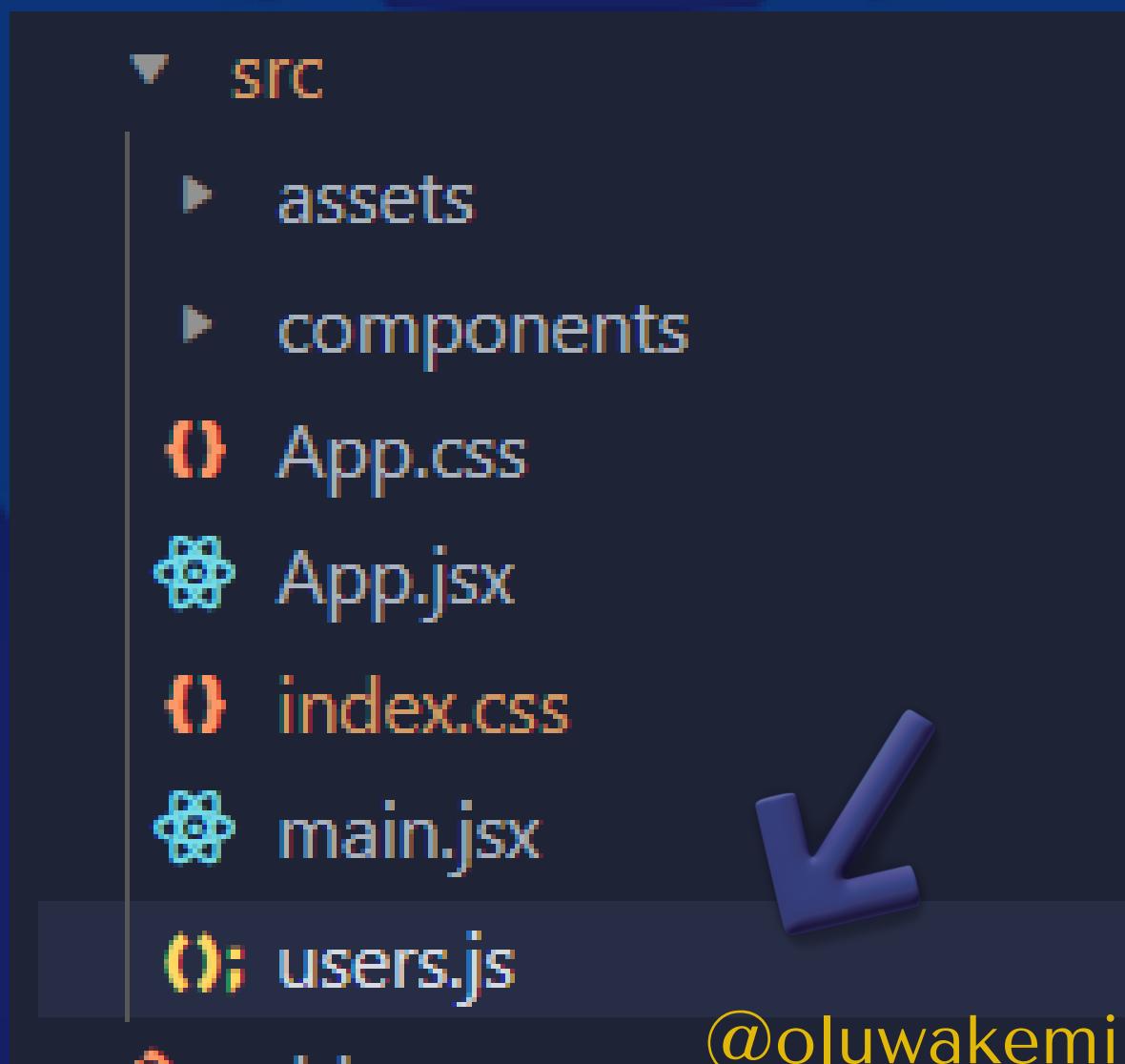
1. Rendering Data from Local file

Let's say you're fetching data from a local data file in your application, and rendering a list dynamically. Here's an example where we fetch user data and render a list of usernames:

Here's an example where we fetch user data and renders it as a list:

Now let's do this step by step:

Step 1: Create a file named “users.js” directly in your ‘src’ folder like this



This file will serve as our data storage file. Now let's populate the data file:

```
1 export const users = [ use your own image
2   {
3     id: 1,           path
4     img: "/images/female.webp",
5     name: "Micheal Fin",
6     role: "Frontend Developer",
7   },
8   {
9     id: 2,
10    img: "/images/male.webp",
11    name: "John Smith",
12    role: "Backend Developer",
13  },
14  {
15    id: 3,
16    img: "/images/female.webp",
17    name: "Alex Johnson",
18    role: "UI/UX Designer",
19  },
20 ];
21
```

This file contains the data for 3 users. Make sure to add some images in a folder (usually assets) and use the path as the value for **img**.

Next is to create a component that renders this data as a list. Name of component is “UserList.jsx”

This component will be rendered as a card for each user:

```
1 import users from "../users";
2
3 function UserList() {
4   return (
5     <>
6       {users.map((user) => (
7         <div key={user.id} className="container">
8           <div className="wrapper">
9             <img
10               src={user.img}
11               alt="user-image"
12               title="user"
13               className="avatar"
14             />
15             <h2>{user.name}</h2>
16             <p>{user.role}</p>
17             </div>
18           </div>
19         ))}
20       </>
21     );
22   }
23
24 export default UserList;
```

What the **UserList** component does:

- The component imports a **list of users** from our data file and uses the **.map()** function to iterate over each user in the array. For each user, it dynamically creates a new **div** element that displays their details.
- Inside each **div**, the component displays the user's **image** (``), **name** (`<h2>`), and **role** (`<p>`), which are dynamically pulled from the user object. The **user.id** is used as a **unique key** for each div.

Simple styling for our user card: add this to your “index.css” file

```
1 .container {  
2   display: flex;  
3   place-items: center;  
4   gap: 2em;  
5   color: white;  
6   font-size: 1.5em;  
7 }  
8  
9 .wrapper {  
10   background: #1b0112;  
11   padding: 1em;  
12   text-align: center;  
13   flex: 1 1 0%;  
14   border-radius: 5px;  
15 }  
16  
17 .avatar {  
18   max-width: 30%;  
19   margin: 0 auto;  
20   aspect-ratio: 1/1;  
21   border-radius: 50%;  
22   border: 5px solid #9d0b51;  
23   box-shadow: 0 0 10px #9d0b51;  
24 }  
25  
26 p {  
27   font-size: 0.8em;  
28   color: #e14490;  
29 }
```

Now run your application to view your project. You should have this cards in your browser:



Micheal Fin
Frontend Developer



John Smith
Backend Developer



Alex Johnson
UI/UX Designer

If you add “**flex-wrap: wrap**” to the container class, you would have this for responsive view:



Micheal Fin
Frontend Developer



John Smith
Backend Developer



Alex Johnson
UI/UX Designer

2. Rendering a Lists of Components

You can also render lists of custom components instead of just HTML elements.

Let's say we create a **UserCard** component and use it to display a card for each user:



```
1 function UserCard({ img, name, role }) {  
2   return (  
3     <div className="wrapper">  
4       <img  
5         src={img}  
6         alt={`${name}'s profile`}  
7         title={name}  
8         className="avatar"  
9       />  
10      <h2>{name}</h2>  
11      <p>{role}</p>  
12    </div>  
13  );  
14 }  
15  
16 export default UserCard;
```

- **UserCard** takes **img**, **name**, and **role** as props.
- It renders a card with an image, name, and role, using these props.

We can now use the **UserCard** component to render the data from our local file (**user.js**):



```
1 import { users } from '../users'; // Import the users data
2 import UserCard from './UserCard'; // Import the UserCard component
3
4 function UserList() {
5   return (
6     <div className="container">
7       {users.map((user) => (
8         <UserCard key={user.id} ←
9           img={user.img}
10          name={user.name}
11          role={user.role} ←
12        />
13      )));
14    </div>
15  );
16}
17
18
19 export default UserList;
```

UserCard component
is used to render each
user detail

mapped user details

- The **UserList** component imports **UserCard** and uses it inside the **map()** method to render each user.
- The **key prop** is set on **UserCard** to uniquely identify each card.

Handling Conditional Rendering with Lists

Sometimes, you might want to conditionally render a list based on certain conditions. For example, you can check if a list is empty before rendering the UI.

From our example:

```
● ● ●

1 import { users } from './users'; // Import the users data
2
3 function UserList() {
4   return (
5     <div className="container">
6       {users.length === 0 ? (
7         // Display this message if the users array is empty
8         <p>No users available</p>
9       ) : (
10         users.map(user => (
11           <div key={user.id} className="wrapper">
12             <img
13               src={user.img}
14               alt={`${user.name}'s profile`}
15               title={user.name}
16               className="avatar"
17             />
18             <h2>{user.name}</h2>
19             <p>{user.role}</p>
20           </div>
21         ))
22       )}
23     </div>
24   );
25 }
26
27 export default UserList;
28
```

In the UserList,

- **users.length === 0** is used to check if the users array is empty.
- If the condition is true (i.e., the array is empty), the `<p>` element with the message "**No users available**" is rendered.
- If the array has data, the **map()** function is used to iterate over each user and render their details.

Best Practices For Using List and Keys In React

1. Use Unique Values: Keys should be unique among siblings. Often, you'll use a **unique ID** from your data as a key, like **user.id**. Avoid using the **array index** as a key, if possible, as this can lead to performance issues and rendering bugs, especially if the list order changes.

Bad Example:



avoid using index as a key value

```
1 {users.map((user, index) => <li key={index}>{user}</li>)}
```

Using the index might work for static lists, but if the order of items changes, React will struggle to track which item is which, potentially causing rendering issues.

2. Avoid Random Keys: Don't generate random values for keys (e.g., using `Math.random()`). React relies on the stability of keys for efficient updates. Using a random key will cause React to re-render the component unnecessarily on each render cycle.

3. Keys Only Need to Be Unique Among Siblings: You don't need globally unique keys across your application; they just need to be unique within their parent container.



I hope you found this material
useful and helpful.

Remember to:

Like

Save for future reference

&

Share with your network, be
helpful to someone 

Hi There!

Thank you for reading through
Did you enjoy this knowledge?

 Follow my LinkedIn page for more work-life balancing and Coding tips.



LinkedIn: Oluwakemi Oluwadahunsi