Day 11

React Lists and Keys

A Comprehensive Guide

@oluwakemi Oluwadahunsi

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 <li> elements or cards) based on that array.

@oluwakemi Oluwadahunsi

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:

```jsx
const users = ['John', 'Jane', 'Mark'];

function UserList() {
  return (
    <ul>
      {users.map(user => <li key={user}>{user}</li>)}
    </ul>
  );
}

export default UserList;
```

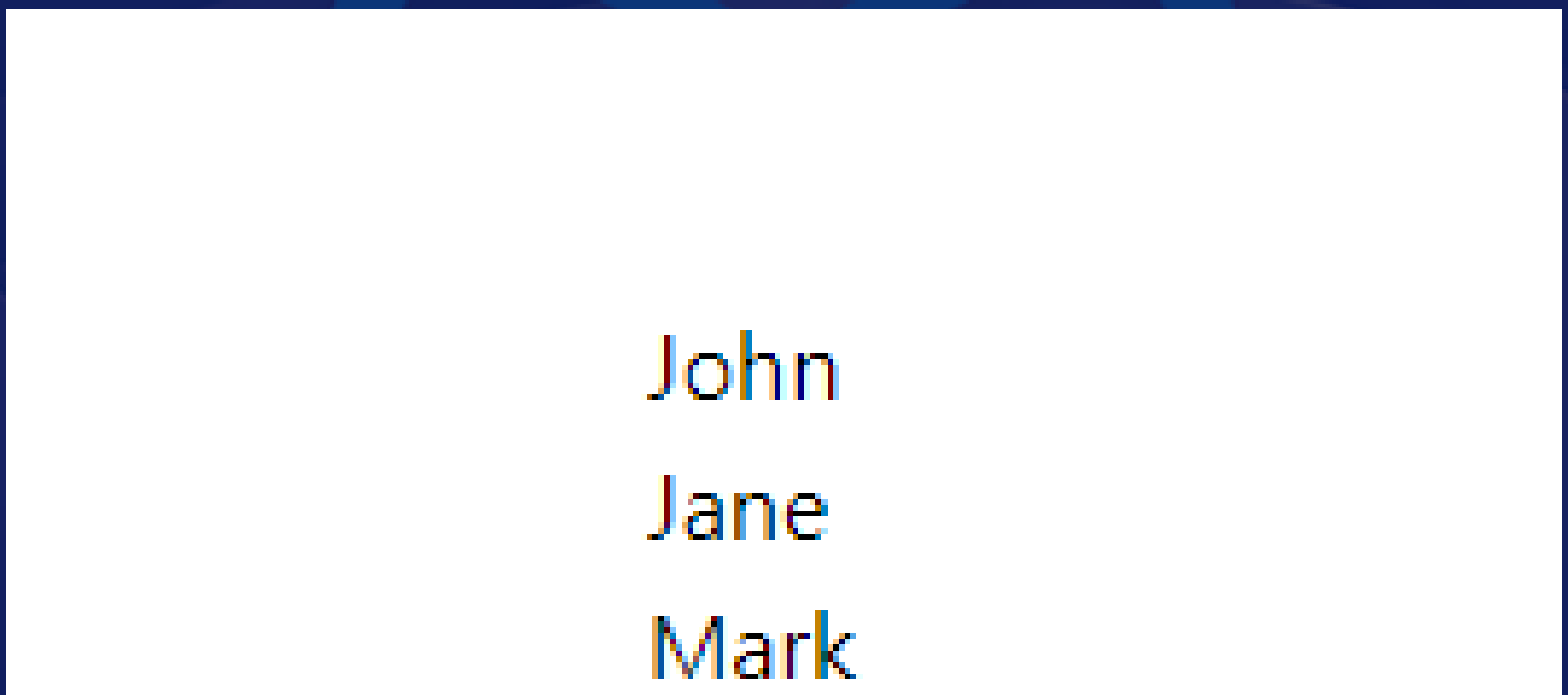
the users array is accessed using .map()

users variable called

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

@oluwakemi Oluwadahunsi

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



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).

@oluwakemi Oluwadahunsi

For Example:

```jsx
const users = [
  { id: 1, name: 'John' },
  { id: 2, name: 'Jane' },
  { id: 3, name: 'Mark' }
];

function UserList() {
  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}

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.

@oluwakemi Oluwadahunsi

Dynamic Lists Example

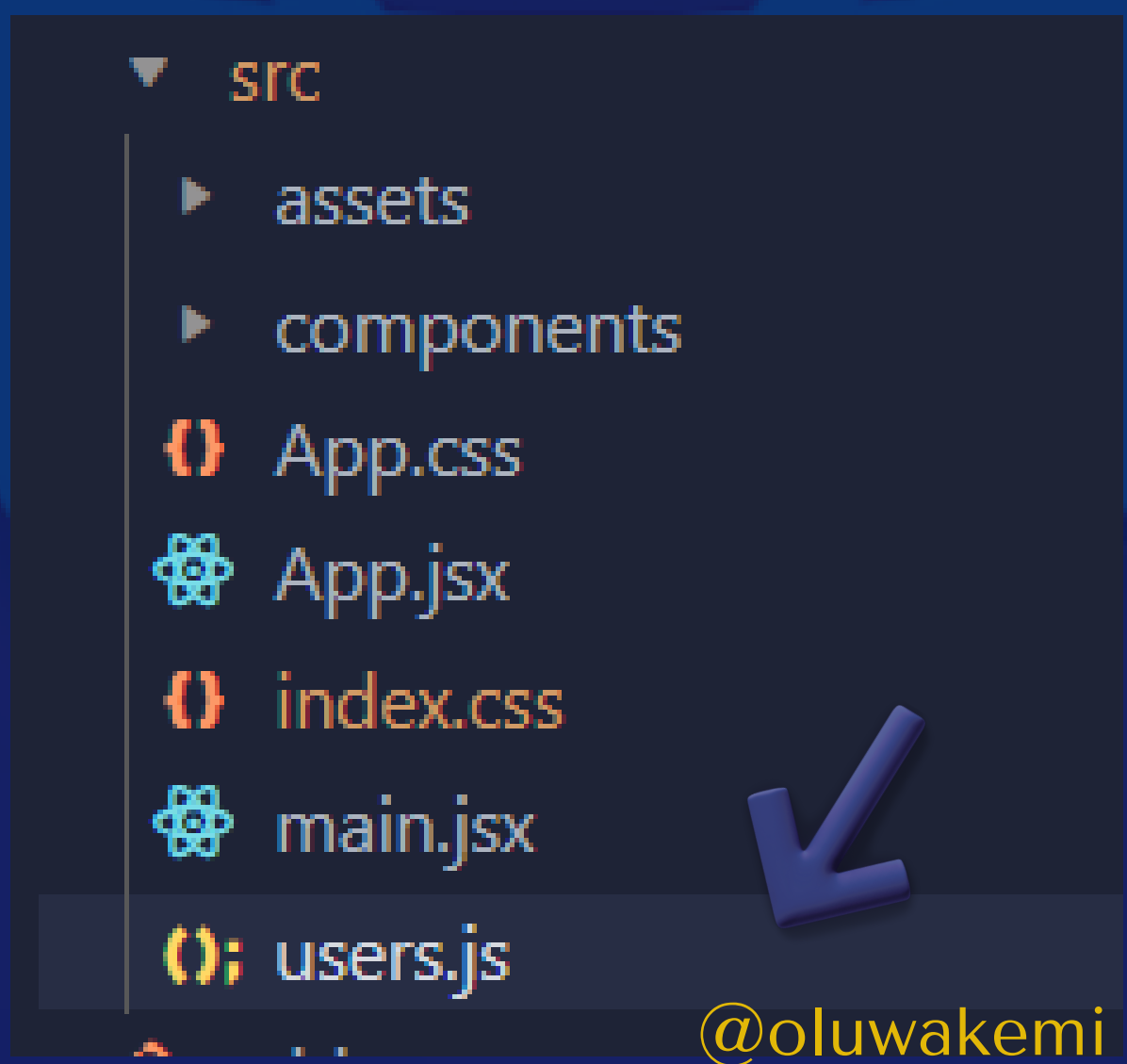
1. Rendering Data from Local file

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:

```
export const users = [
  {
    id: 1,
    img: "/images/female.webp",
    name: "Micheal Fin",
    role: "Frontend Developer",
  },
  {
    id: 2,
    img: "/images/male.webp",
    name: "John Smith",
    role: "Backend Developer",
  },
  {
    id: 3,
    img: "/images/female.webp",
    name: "Alex Johnson",
    role: "UI/UX Designer",
  },
];
```

use your own image path

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**.

@oluwakemi Oluwadahunsi

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

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

```jsx
import users from "../users";

function UserList() {
  return (
    <>
      {users.map((user) => (
        <div key={user.id} className="container">
          <div className="wrapper">
            <img
              src={user.img}
              alt="user-image"
              title="user"
              className="avatar"
            />
            <h2>{user.name}</h2>
            <p>{user.role}</p>
          </div>
        </div>
      ))}
    </>
  );
}

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** (<img>), **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.

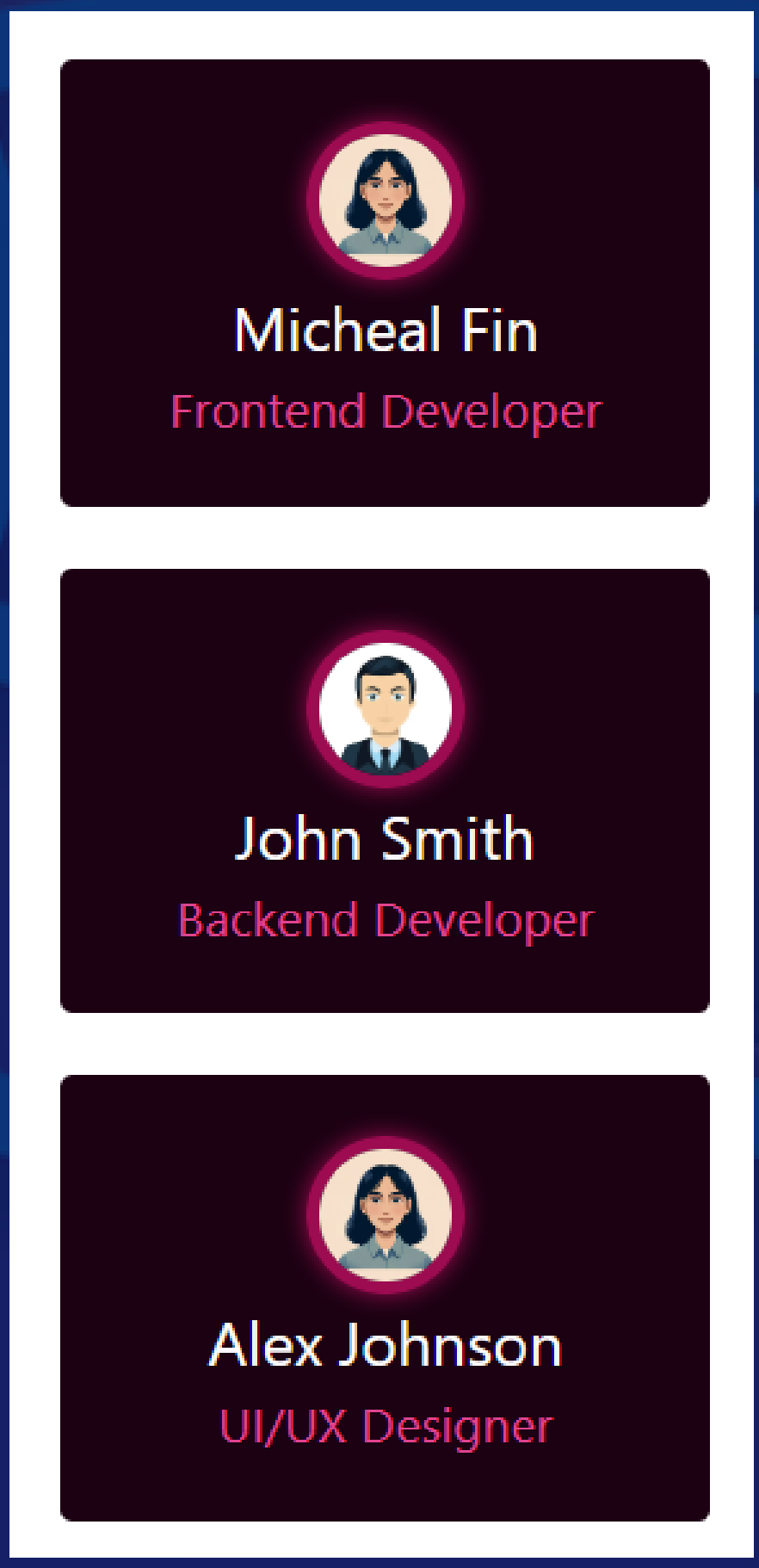
@oluwakemi Oluwadahunsi

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

```css
.container {
  display: flex;
  place-items: center;
  gap: 2em;
  color: white;
  font-size: 1.5em;
}

.wrapper {
    background: #1b0112;
    padding: 1em;
    text-align: center;
    flex: 1 1 0%;
    border-radius: 5px;
}

.avatar {
  max-width: 30%;
  margin: 0 auto;
  aspect-ratio: 1/1;
  border-radius: 50%;
  border: 5px solid #9d0b51;
  box-shadow: 0 0 10px #9d0b51;
}

p {
  font-size: 0.8em;
  color: #e14490;
}
```

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

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



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:

```jsx
function UserCard({ img, name, role }) {
  return (
    <div className="wrapper">
      <img
        src={img}
        alt={`${name}'s profile`}
        title={name}
        className="avatar"
      />
      <h2>{name}</h2>
      <p>{role}</p>
    </div>
  );
}

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**):

```jsx
import { users } from '../users'; // Import the users data
import UserCard from './UserCard'; // Import the UserCard component

function UserList() {
  return (
    <div className="container">
    {users.map((user) => (
          <UserCard
            key={user.id}
            img={user.img}
            name={user.name}
            role={user.role}
          />
        ))}
    </div>
  );
}

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.

@oluwakemi Oluwadahunsi

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:

```jsx
import { users } from '../users'; // Import the users data

function UserList() {
  return (
    <div className="container">
      {users.length === 0 ? (
    // Display this message if the users array is empty
        <p>No users available</p>
      ) : (
        users.map(user => (
          <div key={user.id} className="wrapper">
            <img
              src={user.img}
              alt={`${user.name}'s profile`}
              title={user.name}
              className="avatar"
            />
            <h2>{user.name}</h2>
            <p>{user.role}</p>
          </div>
        ))
      )}
    </div>
  );
}

export default UserList;
```

@oluwakemi Oluwadahunsi

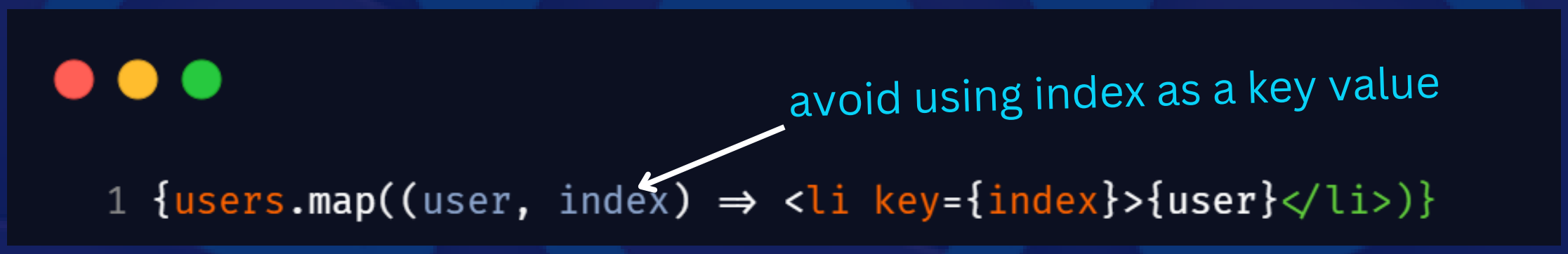
In the UserList,

- **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:



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.

@oluwakemi Oluwadahunsi

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.

@oluwakemi Oluwadahunsi

I hope you found this material useful and helpful.

Remember to:

Like

Save for future reference

&

Share with your network, be helpful to someone 👌

@oluwakemi Oluwadahunsi

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

kodemaven-portfolio.vercel.app