Routers and Lists

Topics covered:

- What are React routers?
  - Installing React Router
- Types of routers
- Nesting (routers inside routers)
- Lists and keys in ReactJs

1. **React routers:**

   React Router is a standard library for React routing. It allows navigating between views of different components in a React Application, changes the browser URL, and keeps the UI in sync with the URL.

   To understand how the React Router works, let's build a simple React application. The application will have three parts: the **home** component, the **about** a component, and the **contact** component. To move between these components, we will use React Router.

- **Installing React Router:**

  You may install React Router in your React application using npm. To install the Router in your React application, follow the steps below:

  Step 1: Navigate to your project's directory by using the command:

  ```
  cd project-dir-name
  ```

  Step 2: Run the following command to install the React Router:

  ```
  npm install react-router-dom
  ```

  Add react-router-dom components to your React application after installing it.

  Adding React Router Components: The major React Router components are:

  BrowserRouter is a router solution that makes use of the HTML5 history API (pushState, replaceState, and the popstate event) to keep your UI in sync with the URL. All of the other components are stored in the parent component.

  Routes: This is a new component introduced in v6 as well as an improvement to the component.

  The following are the main advantages of Routes vs Switches:

  Instead of being traversed in order,

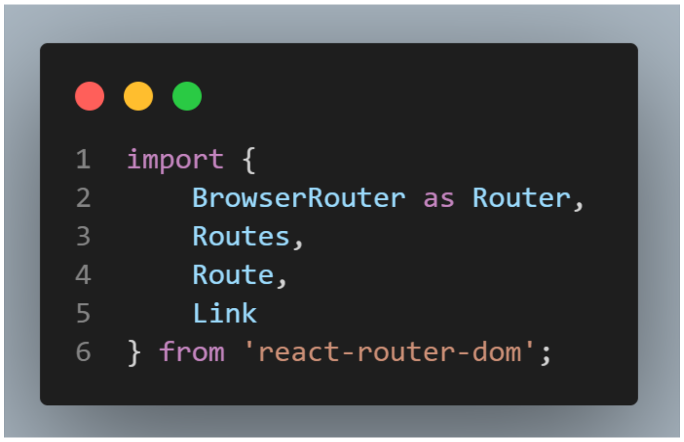
  - relative s and s

- Routes are picked based on the best match.

Route: A route is a conditionally displayed component that displays some UI when its path matches the current URL.

Link: The link component is used to create linkages to different routes and to provide application navigation. It functions similarly to the HTML anchor tag.

To include React Router components in your application, enter your project directory in your preferred editor and navigate to the app.js file. Now, in app.js, paste the code below.



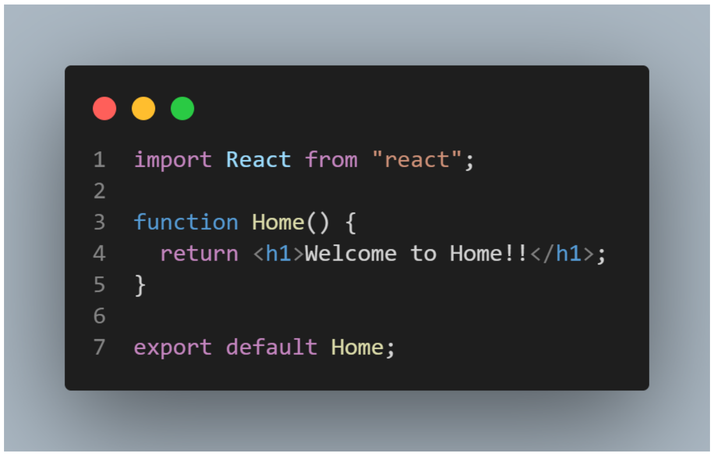
```
import {
    BrowserRouter as Router,
    Routes,
    Route,
    Link
} from 'react-router-dom';
```

Note: BrowserRouter is also known as Router.

Using React Router: Before we begin using React Router, we'll need to develop a few components in our react application. Create a component folder inside the src folder in your project directory, and then add three files to the component folder: home.js, about.js, and contact.js.

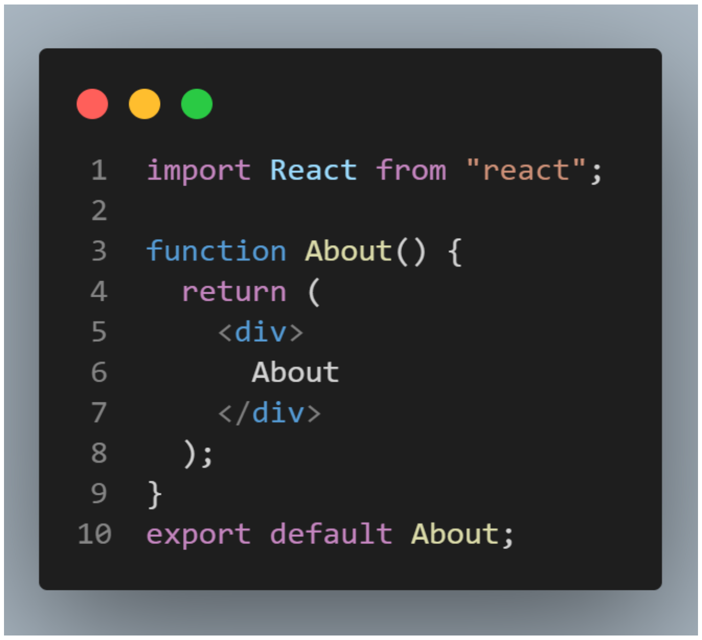
Let's put some code into our three components:

Home.js:

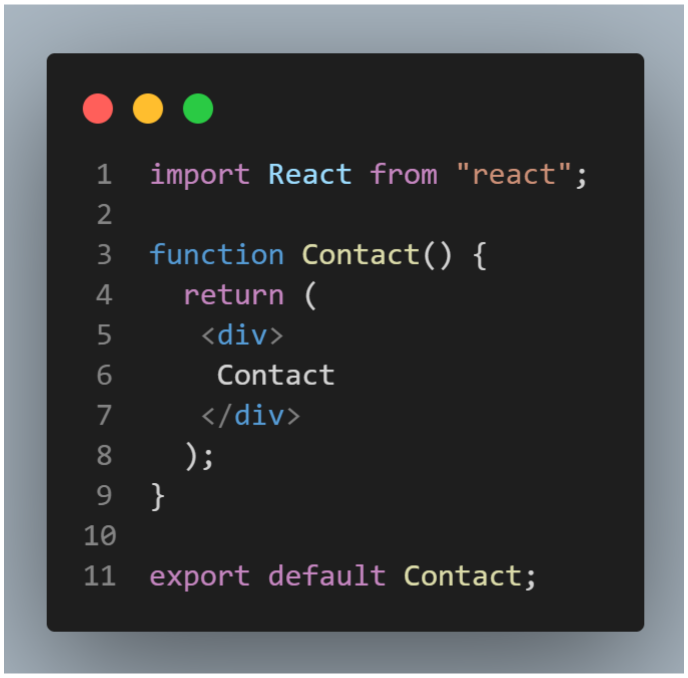


```
import React from "react";

function Home() {
  return <h1>Welcome to Home!!</h1>;
}

export default Home;
```

About.js:


```jsx
import React from "react";

function About() {
  return (
    <div>
      About
    </div>
  );
}
export default About;
```

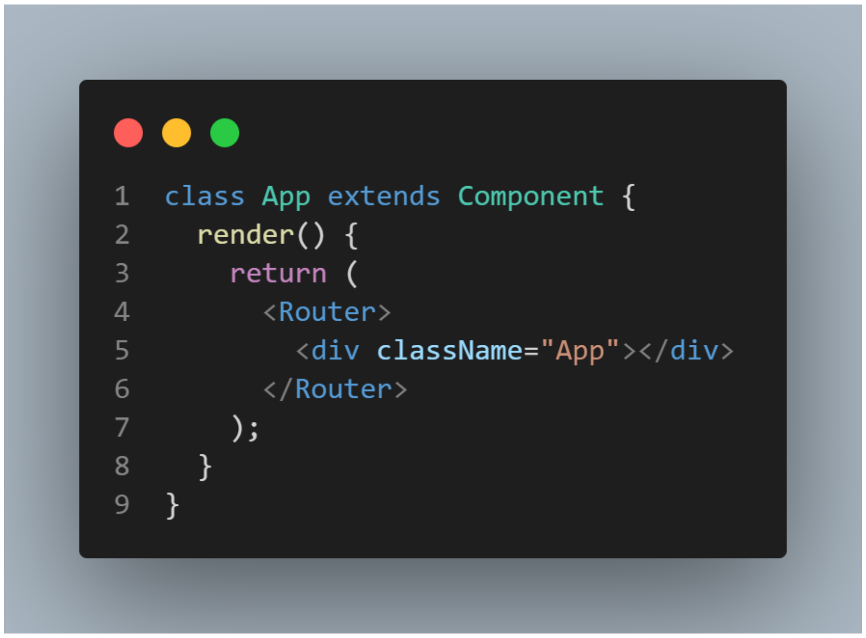
Contact.js:


```jsx
import React from "react";

function Contact() {
  return (
   <div>
    Contact
   </div>
  );
}

export default Contact;
```

Now, let's add some React Router components to the app:

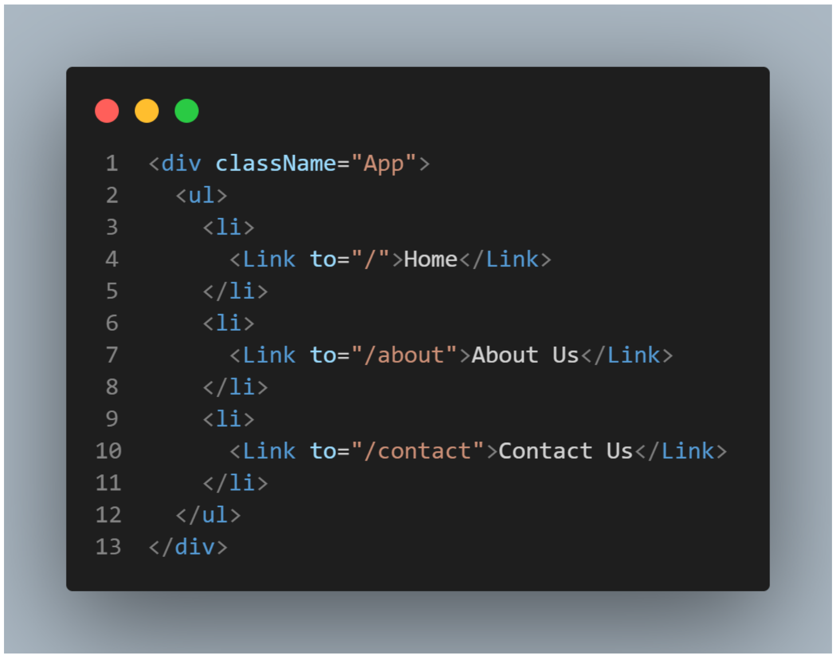
Add BrowserRouter, aliased as Router, to your appjs file to wrap all the other components.

BrowserRouter is a parent component that can only have one child.



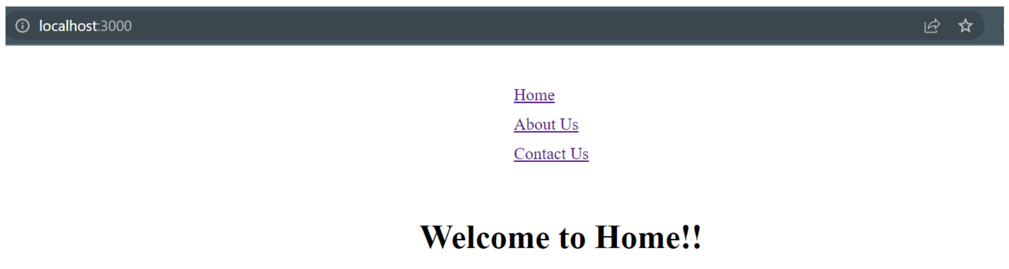
```
class App extends Component {
  render() {
    return (
      <Router>
        <div className="App"></div>
      </Router>
    );
  }
}
```

Let us now make links to our components. The to prop is used by the link component to indicate the place to which the links should navigate.



```
<div className="App">
  <ul>
    <li>
      <Link to="/">Home</Link>
    </li>
    <li>
      <Link to="/about">About Us</Link>
    </li>
    <li>
      <Link to="/contact">Contact Us</Link>
    </li>
  </ul>
</div>
```

Run your program on the local host now, then click the links you made. The url will adjust in accordance with the value entered for the Link component's properties.


localhost:3000

Home
About Us
Contact Us

Welcome to Home!!

You can now access the various components by clicking on the links. Your application's UI and URL are kept in sync via React Router.

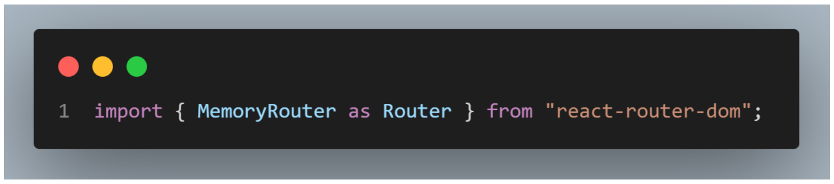
2. **Types of Router:**

   React Router offers three different types of routers based on the portion of the URL that the router will use to monitor the content that the user is attempting to view:

   - Memory Router
   - Browser Router
   - Hash Router

   Memory Router: This type of router retains updated URLs in memory rather than in user browsers. The user is unable to use the back and forward buttons on the browser since it saves the URL history in memory and doesn't read or write to the address bar. Your browser's URL is left unchanged. For testing and non-browser environments like React Native, it is incredibly helpful.

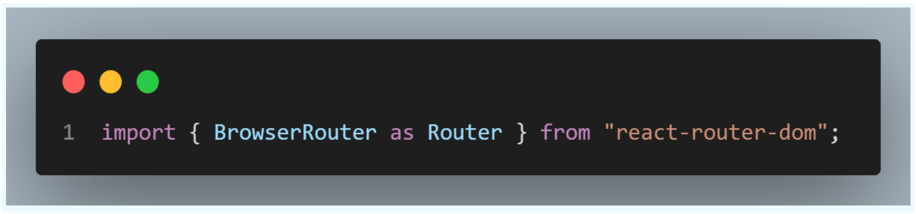
   Syntax:



```
import { MemoryRouter as Router } from "react-router-dom";
```

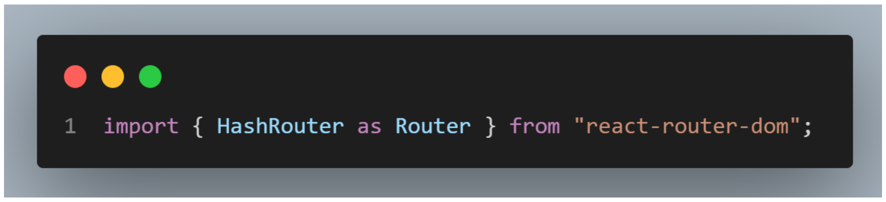
   Browser Router: The browser router makes advantage of the pushState, replaceState, and popState APIs from the HTML 5 history standard to keep your user interface (UI) in sync with the URL. It routes as a normal URL in the browser and expects that the server handles all request URLs (for example, /, /about) and leads to root index.html. It supports legacy browsers that do not implement HTML 5 pushState API by accepting forceRefresh props.

   Syntax:



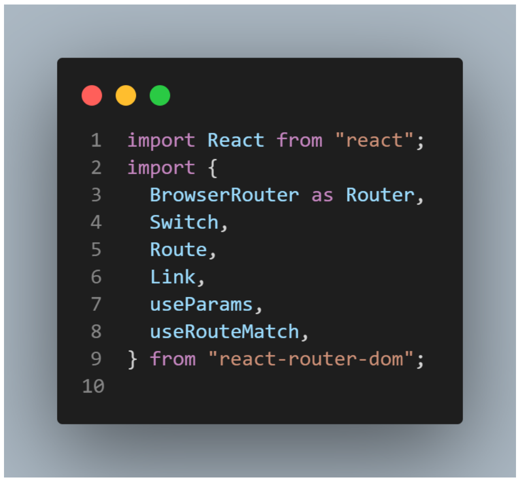
```
import { BrowserRouter as Router } from "react-router-dom";
```

Hash Router: Client-side hash routing is used by hash router. To keep your UI in sync with the URL, it uses the hash component of the URL (i.e. window.location.hash). The server won't handle the hash portion of the URL; instead, it will always send index.html in response to requests and ignore the hash value. The server does not need to be configured in order to handle routes. The purpose of it is to accommodate older browsers, which often do not implement HTML pushState API. When using legacy browsers or when there is no server logic to handle client-side processing, it is quite helpful. The react-router-dom team does not advise using this route.
Syntax:

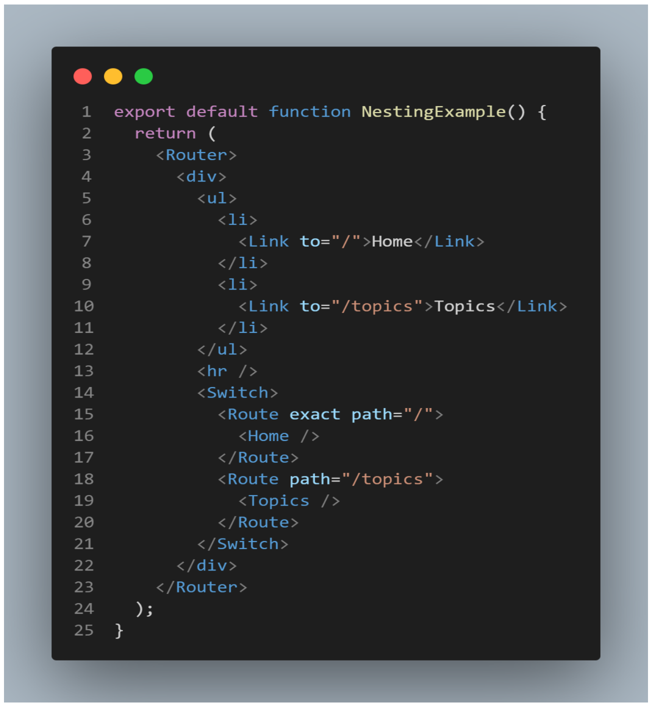


```
import { HashRouter as Router } from "react-router-dom";
```

3. **Nesting:**
    - Routes can be rendered anywhere in the app, including in child elements, as they are standard React components.
    - Because code-splitting a React Router app is the same as code-splitting any other React project, this is useful when it comes time to divide your app into various bundles.

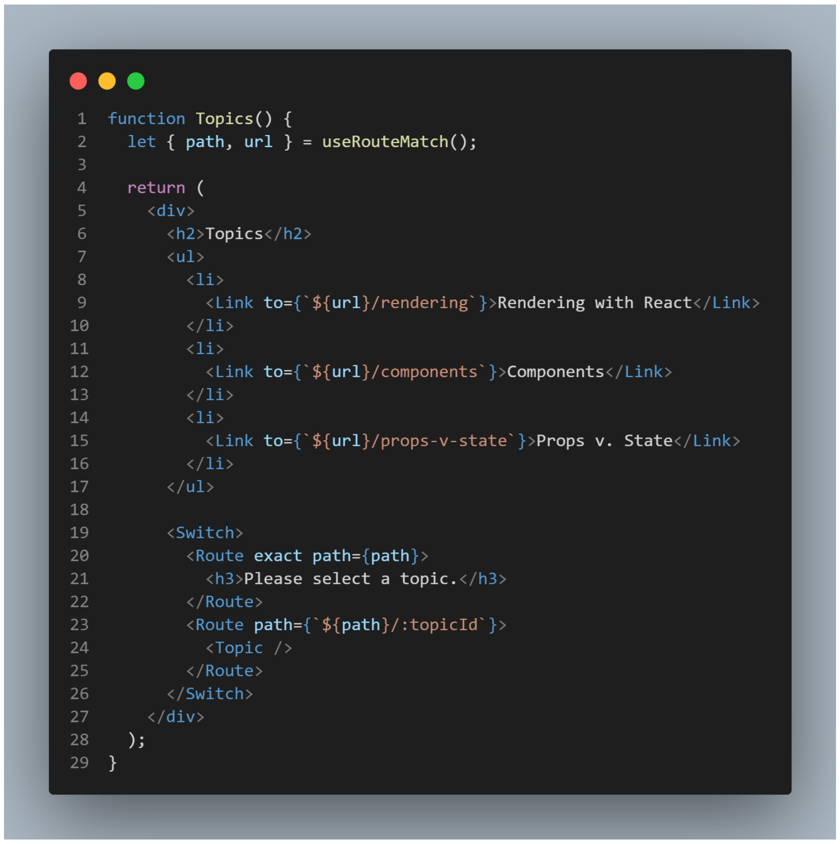


```
import React from "react";
import {
  BrowserRouter as Router,
  Switch,
  Route,
  Link,
  useParams,
  useRouteMatch,
} from "react-router-dom";

```



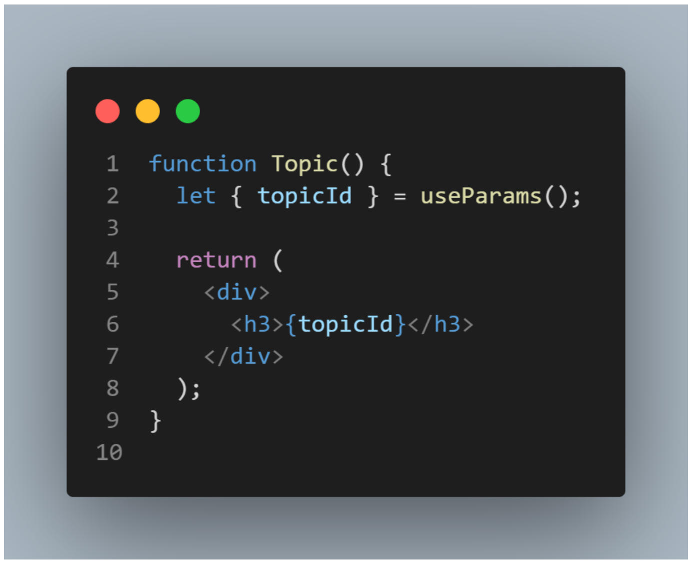
```
export default function NestingExample() {
  return (
    <Router>
      <div>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/topics">Topics</Link>
          </li>
        </ul>
        <hr />
        <Switch>
          <Route exact path="/">
            <Home />
          </Route>
          <Route path="/topics">
            <Topics />
          </Route>
        </Switch>
      </div>
    </Router>
  );
}
```

- The 'path' allows us to create relative <Route> pathways to the parent route, whereas the 'url' enables us to create relative links.

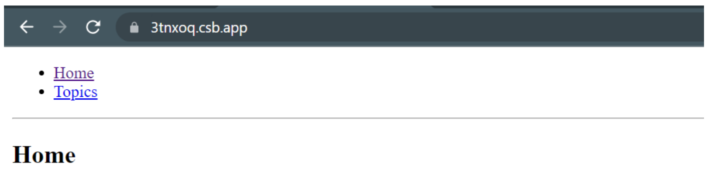


```
function Topics() {
  let { path, url } = useRouteMatch();

  return (
    <div>
      <h2>Topics</h2>
      <ul>
        <li>
          <Link to={`${url}/rendering`}>Rendering with React</Link>
        </li>
        <li>
          <Link to={`${url}/components`}>Components</Link>
        </li>
        <li>
          <Link to={`${url}/props-v-state`}>Props v. State</Link>
        </li>
      </ul>

      <Switch>
        <Route exact path={path}>
          <h3>Please select a topic.</h3>
        </Route>
        <Route path={`${path}/:topicId`}>
          <Topic />
        </Route>
      </Switch>
    </div>
  );
}
```

- This component was produced by a route with the path '/topics/:topicId'. The URL's ':topicId' component denotes a placeholder that can be obtained by calling 'useParams()'.



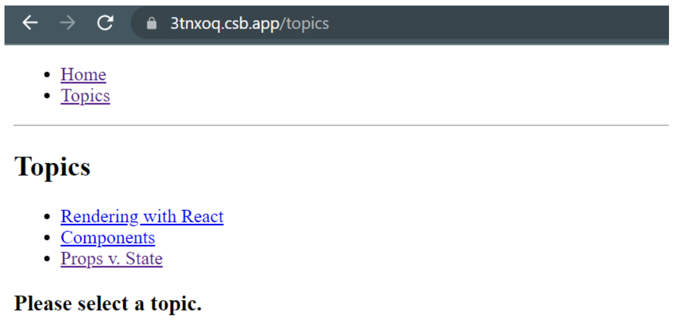
```
function Topic() {
  let { topicId } = useParams();

  return (
    <div>
      <h3>{topicId}</h3>
    </div>
  );
}

```



3tnxoq.csb.app

- Home
- Topics

Home



3tnxoq.csb.app/topics

- Home
- Topics

Topics

- Rendering with React
- Components
- Props v. State

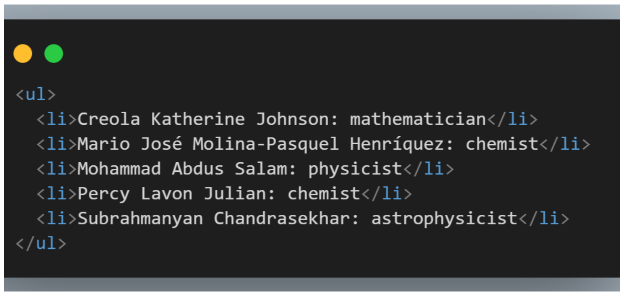
Please select a topic.

4. **Lists and keys:**

a. **Rendering List:**

- Data rendering from arrays

Assume you have a content list.

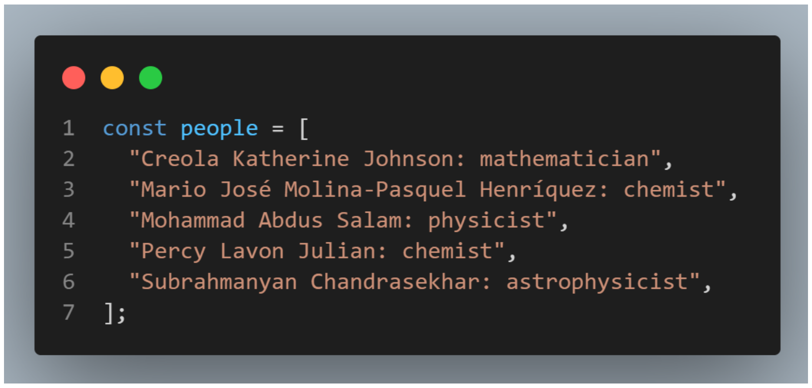


```html
<ul>
  <li>Creola Katherine Johnson: mathematician</li>
  <li>Mario José Molina-Pasquel Henríquez: chemist</li>
  <li>Mohammad Abdus Salam: physicist</li>
  <li>Percy Lavon Julian: chemist</li>
  <li>Subrahmanyan Chandrasekhar: astrophysicist</li>
</ul>
```

The only distinction between those list items is their content, or data. When creating interfaces, you may frequently need to display multiple instances of the same component using different data: from lists of comments to galleries of profile photographs. As these cases, you can save the data in JavaScript objects and arrays and then use techniques like map() and filter() to generate lists of components.

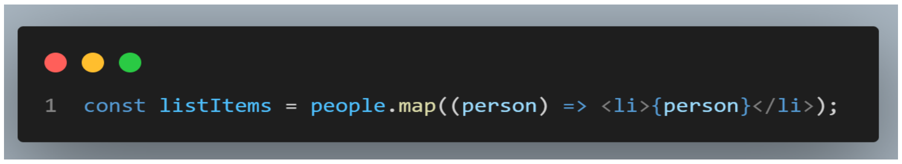
Here's a quick illustration of how to generate a list of objects from an array:

1.) Put the data in an array:



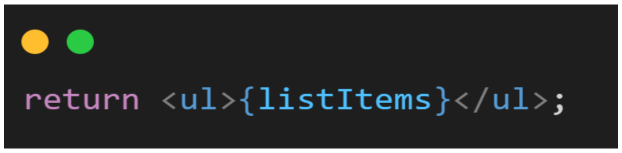
```javascript
const people = [
  "Creola Katherine Johnson: mathematician",
  "Mario José Molina-Pasquel Henríquez: chemist",
  "Mohammad Abdus Salam: physicist",
  "Percy Lavon Julian: chemist",
  "Subrahmanyan Chandrasekhar: astrophysicist",
];
```

2.) Map the members of the persons array to a new array of JSX nodes, listItems:



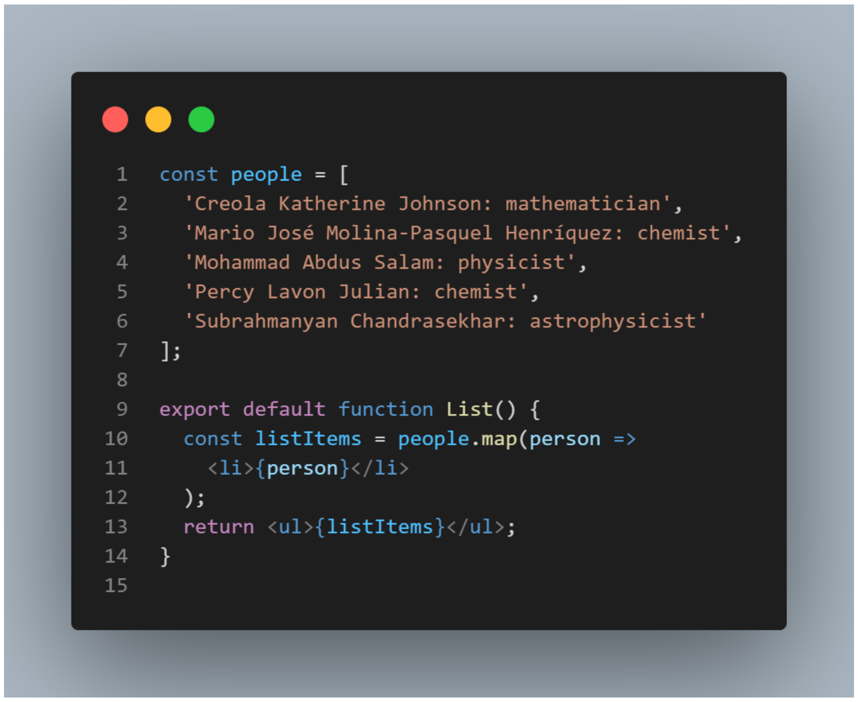
```javascript
const listItems = people.map((person) => <li>{person}</li>);
```

3.) Return listItems wrapped in an <ul> from your component:

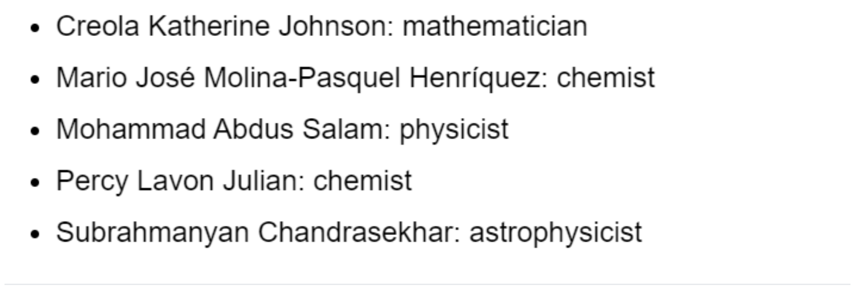


```
return <ul>{listItems}</ul>;
```

4.) The end result is as follows:



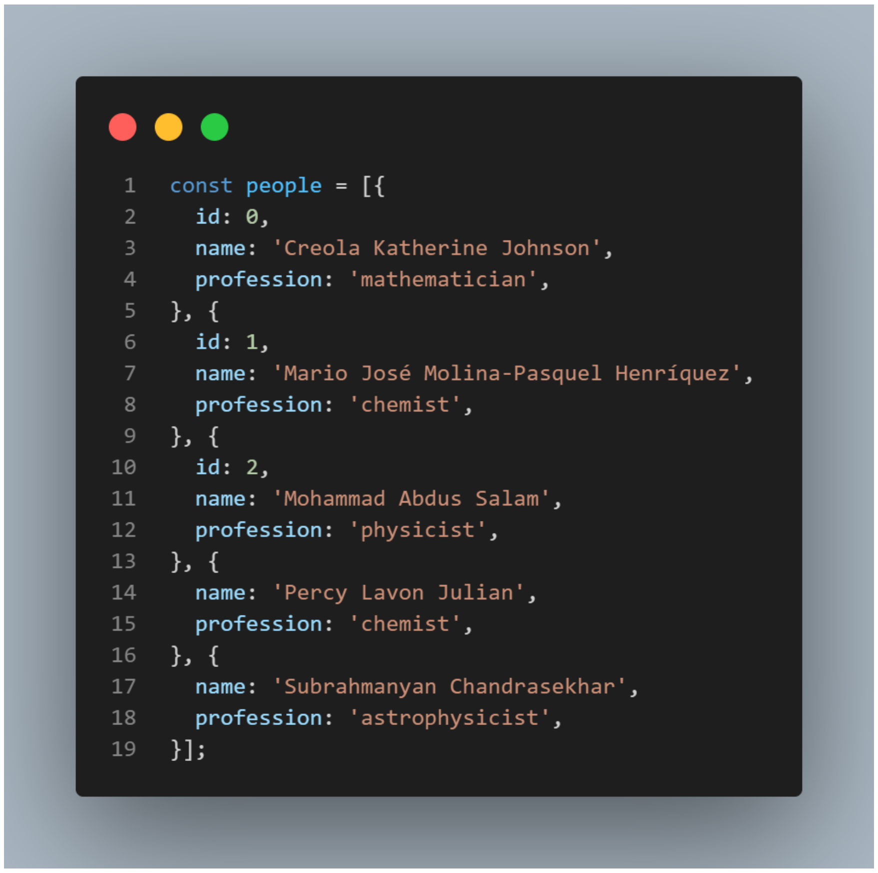
```
const people = [
  'Creola Katherine Johnson: mathematician',
  'Mario José Molina-Pasquel Henríquez: chemist',
  'Mohammad Abdus Salam: physicist',
  'Percy Lavon Julian: chemist',
  'Subrahmanyan Chandrasekhar: astrophysicist'
];

export default function List() {
  const listItems = people.map(person =>
    <li>{person}</li>
  );
  return <ul>{listItems}</ul>;
}
```



- Creola Katherine Johnson: mathematician
- Mario José Molina-Pasquel Henríquez: chemist
- Mohammad Abdus Salam: physicist
- Percy Lavon Julian: chemist
- Subrahmanyan Chandrasekhar: astrophysicist

- Filtering objects in arrays

  This data can be further organized.

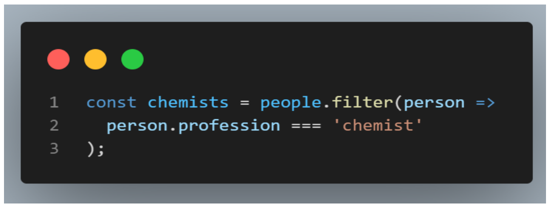


```javascript
const people = [{
  id: 0,
  name: 'Creola Katherine Johnson',
  profession: 'mathematician',
}, {
  id: 1,
  name: 'Mario José Molina-Pasquel Henríquez',
  profession: 'chemist',
}, {
  id: 2,
  name: 'Mohammad Abdus Salam',
  profession: 'physicist',
}, {
  name: 'Percy Lavon Julian',
  profession: 'chemist',
}, {
  name: 'Subrahmanyan Chandrasekhar',
  profession: 'astrophysicist',
}];
```

Assume you wish to only see those whose occupation is chemist. To return only those individuals, utilize JavaScript's filter() function. This method accepts an array of items, runs them through a "test" (a function that returns true or false), and returns a new array containing only the things that passed the test (returned true).

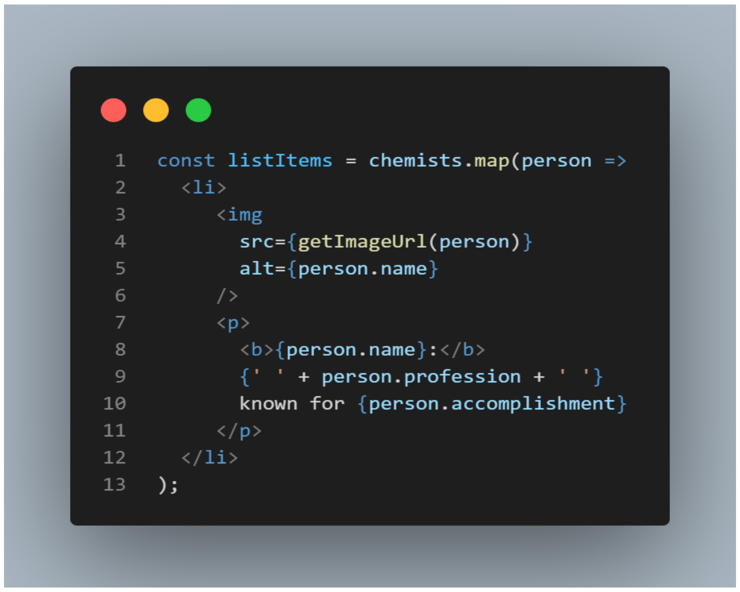
You only want articles with the occupation chemist. The "test" function is(person) => person. profession === 'chemist'. Here's how to do it:

1. You can create a new array of people who are simply "chemists" by using person.profession == 'chemist':



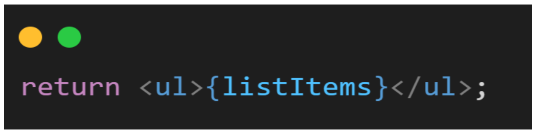
```javascript
const chemists = people.filter(person =>
  person.profession === 'chemist'
);
```

2. Map the chemists next:



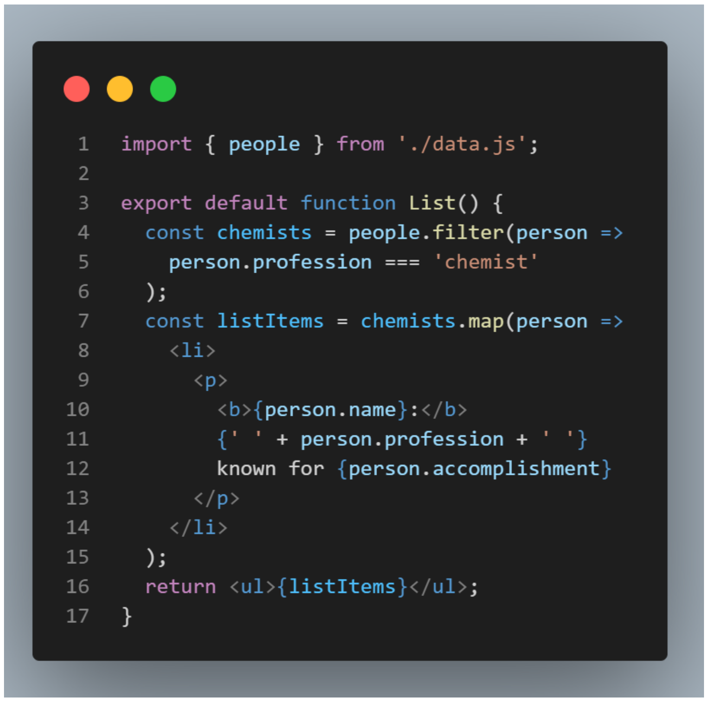
```
const listItems = chemists.map(person =>
  <li>
     <img
       src={getImageUrl(person)}
       alt={person.name}
     />
     <p>
       <b>{person.name}:</b>
       {' ' + person.profession + ' '}
       known for {person.accomplishment}
     </p>
  </li>
);
```

3. Return your component's listItems as a final step:



```
return <ul>{listItems}</ul>;
```

4. The end result is as follows:



```
import { people } from './data.js';

export default function List() {
  const chemists = people.filter(person =>
    person.profession === 'chemist'
  );
  const listItems = chemists.map(person =>
    <li>
      <p>
        <b>{person.name}:</b>
        {' ' + person.profession + ' '}
        known for {person.accomplishment}
      </p>
    </li>
  );
  return <ul>{listItems}</ul>;
}
```

Mario José Molina-Pasquel Henríquez: chemist known for discovery of Arctic ozone hole

Percy Lavon Julian: chemist known for pioneering cortisone drugs, steroids and birth control pills

The code will execute when you run it in your create-react-app, but you will be informed that the list items do not have a "key" specified.

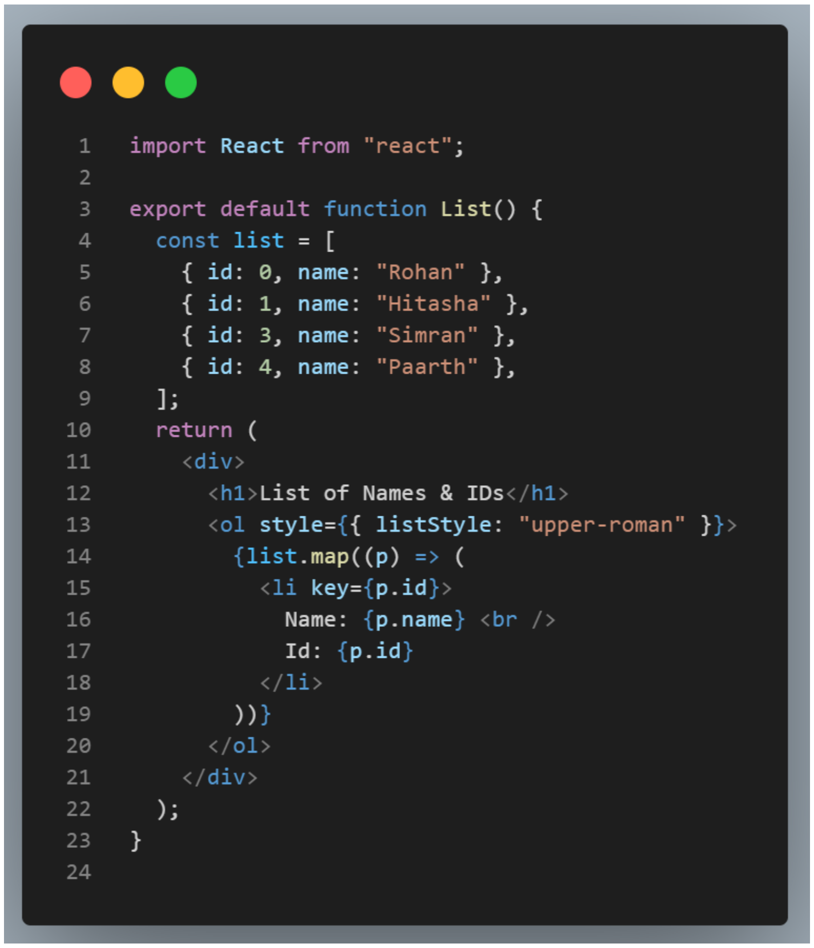
```
Warning: Each child in a list should have a unique “key” prop.
```

```
Warning: Each child in a list should have a unique "key" prop.

Check the render method of `List`. See https://reactjs.org/link/warning-keys for more information.
    at li
    at List
```

b. Key:

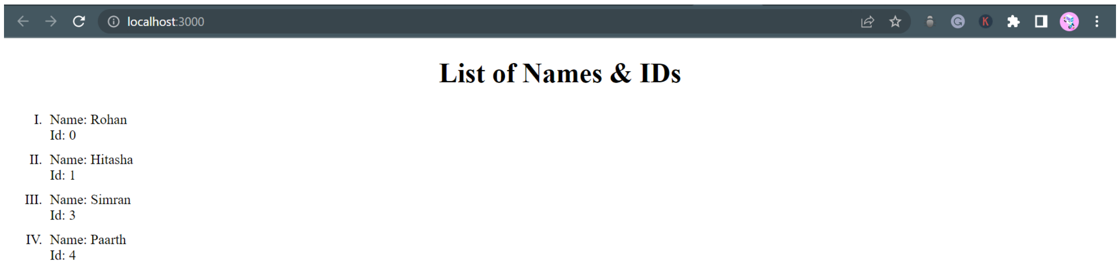
- React uses keys to keep track of elements. In this manner, only the modified or deleted item will be re-rendered and not the full list.
- Each sibling needs their own set of keys. They can, however, be globally duplicated.
- Typically, each item's unique ID serves as the key. The array index might be used as a key as a last resort.

Example:



```jsx
import React from "react";

export default function List() {
  const list = [
    { id: 0, name: "Rohan" },
    { id: 1, name: "Hitasha" },
    { id: 3, name: "Simran" },
    { id: 4, name: "Paarth" },
  ];
  return (
    <div>
      <h1>List of Names & IDs</h1>
      <ol style={{ listStyle: "upper-roman" }}>
        {list.map((p) => (
          <li key={p.id}>
            Name: {p.name} <br />
            Id: {p.id}
          </li>
        ))}
      </ol>
    </div>
  );
}

```

Output:



localhost:3000

```
List of Names & IDs

  I. Name: Rohan
     Id: 0
 II. Name: Hitasha
     Id: 1
III. Name: Simran
     Id: 3
 IV. Name: Paarth
     Id: 4
```