REACT COMPONENT LIFECYCLE WORKSHOP

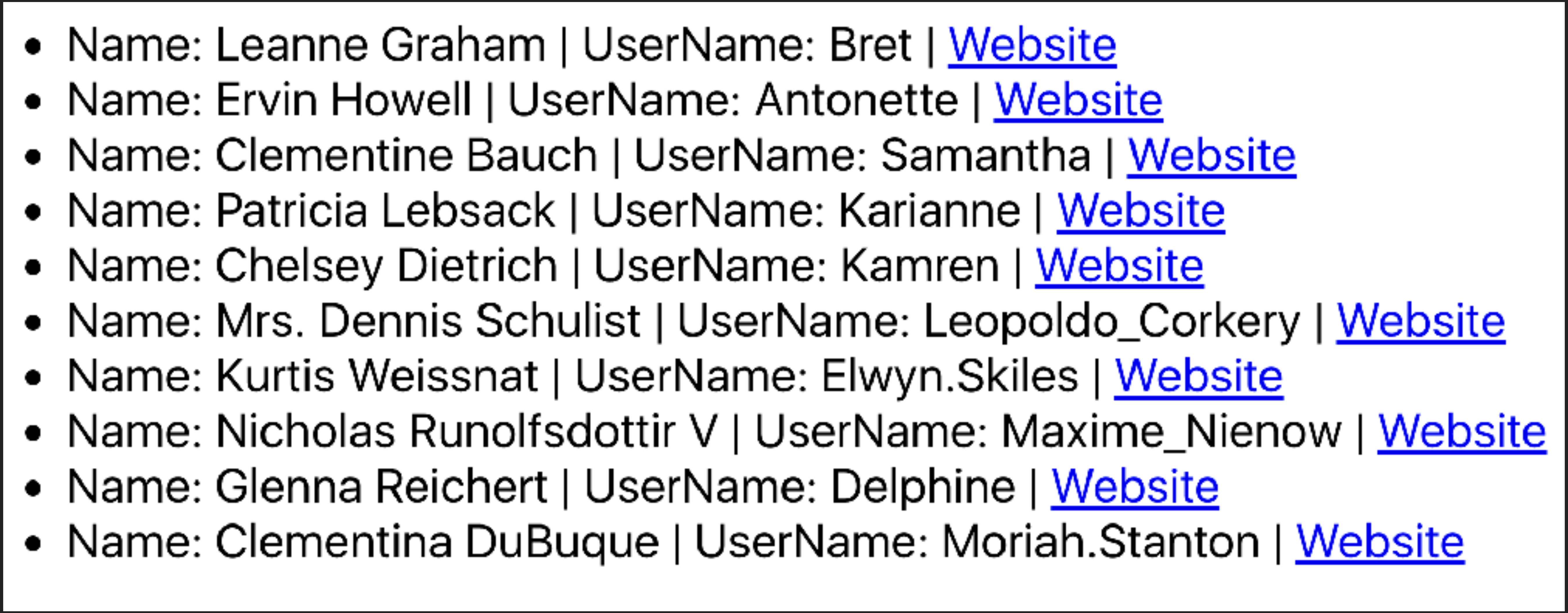
BY KIRK SULLIVAN

CREATE-REACT-APP

- Create a folder named 'React' or if you've already have a directory you want the project to be installed to
- Type - "npx create-react-app testapi" in our console where you want the folder containing the project to be installed into.
- Follow the suggested prompt of running the commands

  ```
  cd testapi
  npm start
  ```
- Now we should see our server running and the react-app boilerplate site being displayed.

We just want a list of the Name Username and a link to each users website



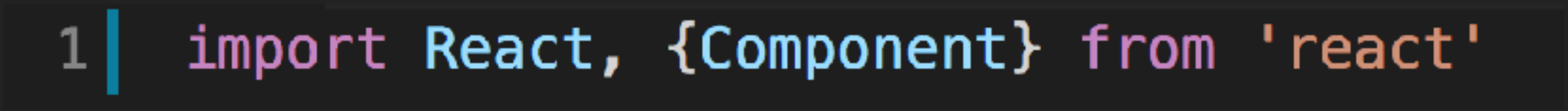
- Name: Leanne Graham | UserName: Bret | Website
- Name: Ervin Howell | UserName: Antonette | Website
- Name: Clementine Bauch | UserName: Samantha | Website
- Name: Patricia Lebsack | UserName: Karianne | Website
- Name: Chelsey Dietrich | UserName: Kamren | Website
- Name: Mrs. Dennis Schulist | UserName: Leopoldo_Corkery | Website
- Name: Kurtis Weissnat | UserName: Elwyn.Skiles | Website
- Name: Nicholas Runolfsdottir V | UserName: Maxime_Nienow | Website
- Name: Glenna Reichert | UserName: Delphine | Website
- Name: Clementina DuBuque | UserName: Moriah.Stanton | Website

- First, think about what type of component that you'll be creating for this workshop.
- Secondly, think about the functionality of this program and what type of methods we will use to accomplish an API call.
- Third, think about how you are going to format this data and how we will display it correctly in JSX.

- For this program we’re going to need to create a stateful class component. This will allow us to create state that we can update/use in our program to represent the data we collect from the API.
- The main functionality of this program is to go out and grab some JSON data on the web and then format it to correctly be displayed to our specifications.

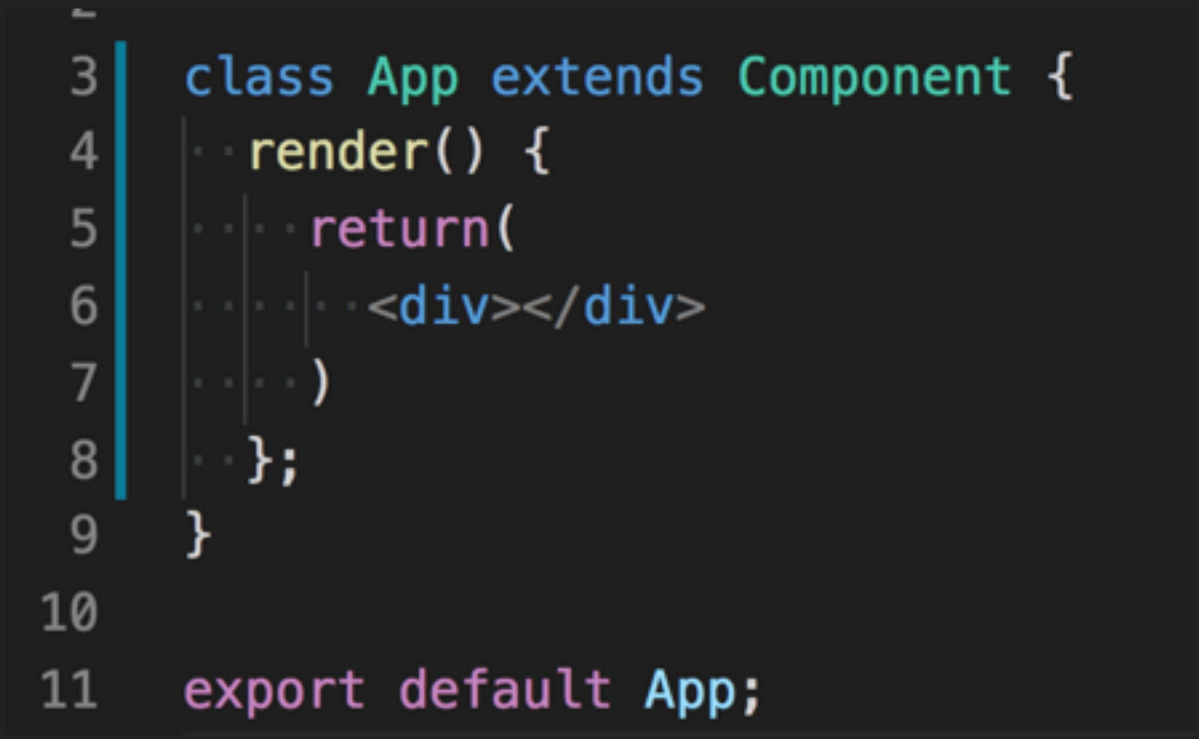
- Lets start coding. First we need to delete everything in app.js file and start from scratch.

Start off by importing your required libraries



```
import React, {Component} from 'react'
```

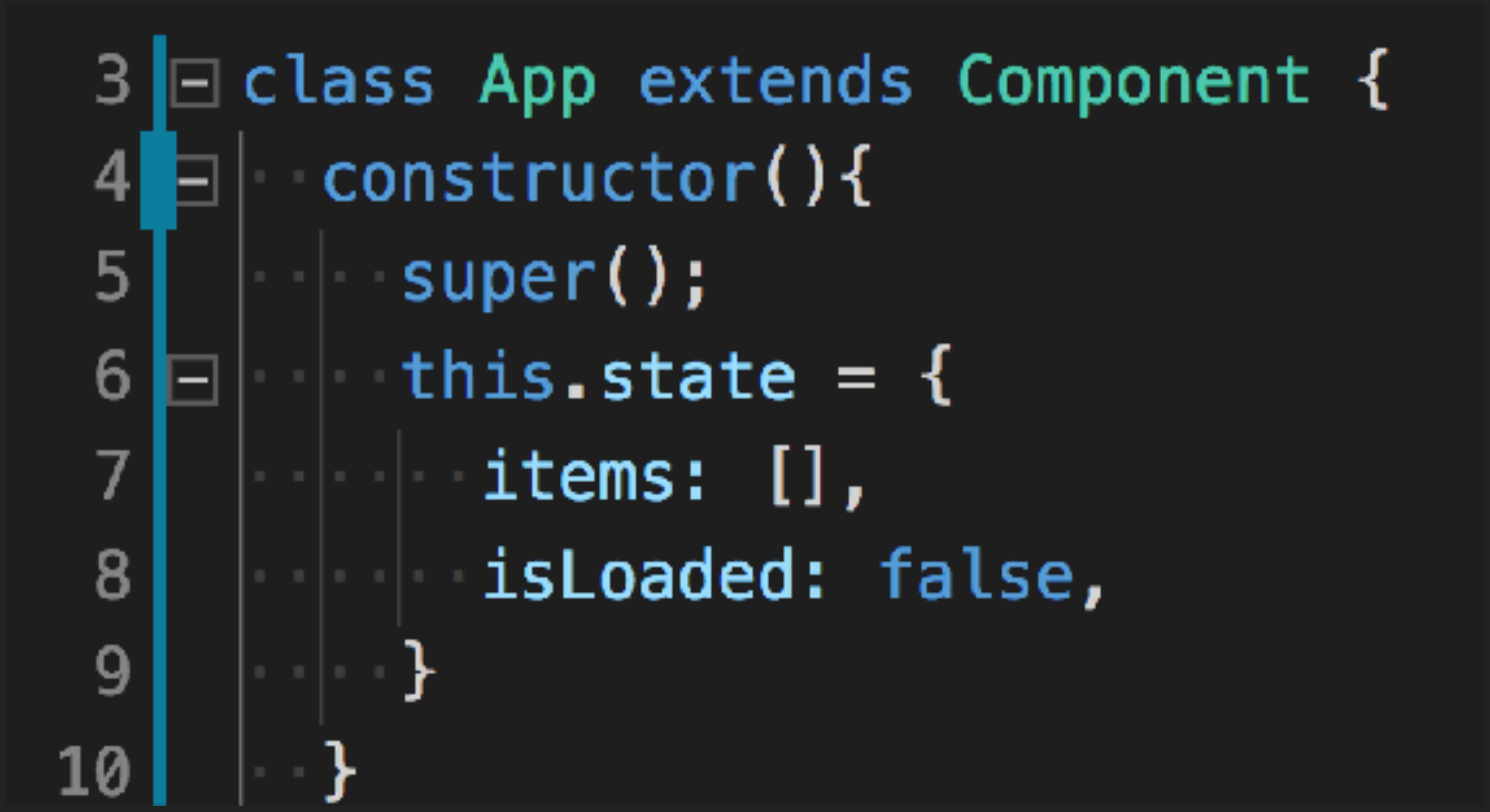
Then build out the base of your component



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

export default App;
```

- Since we know we'll use state for this application let's build our constructor function.



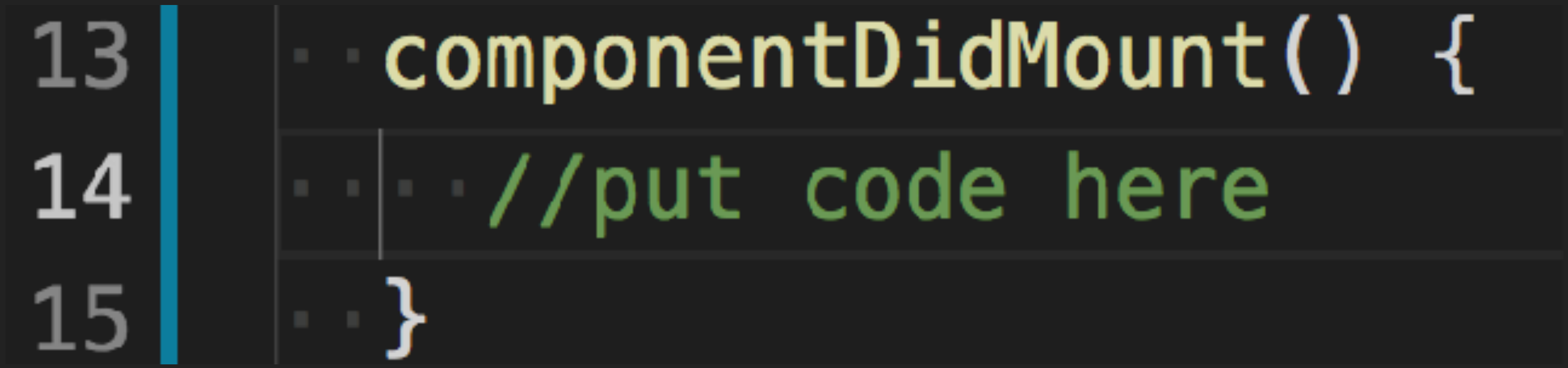
```
class App extends Component {
  constructor(){
    super();
    this.state = {
      items: [],
      isLoaded: false,
    }
  }
```

Items will be initialized to an empty array where we will store the data we retrieve from the API.

isLoaded is a boolean we set to false that we will use later to make sure the data we received is populated

- Now we need to grab the data from an API that contains JSON objects and use a lifecycle method to accomplish this task.
- Think about what lifecycle method will we want to use for an API call.

- To accomplish a API call we want to use the lifecycle method componentDidMount() … This will allow us to attach the data we get from the API call to the state. Remember, when we change the state the component render() method will execute again and show the changes made to the DOM.

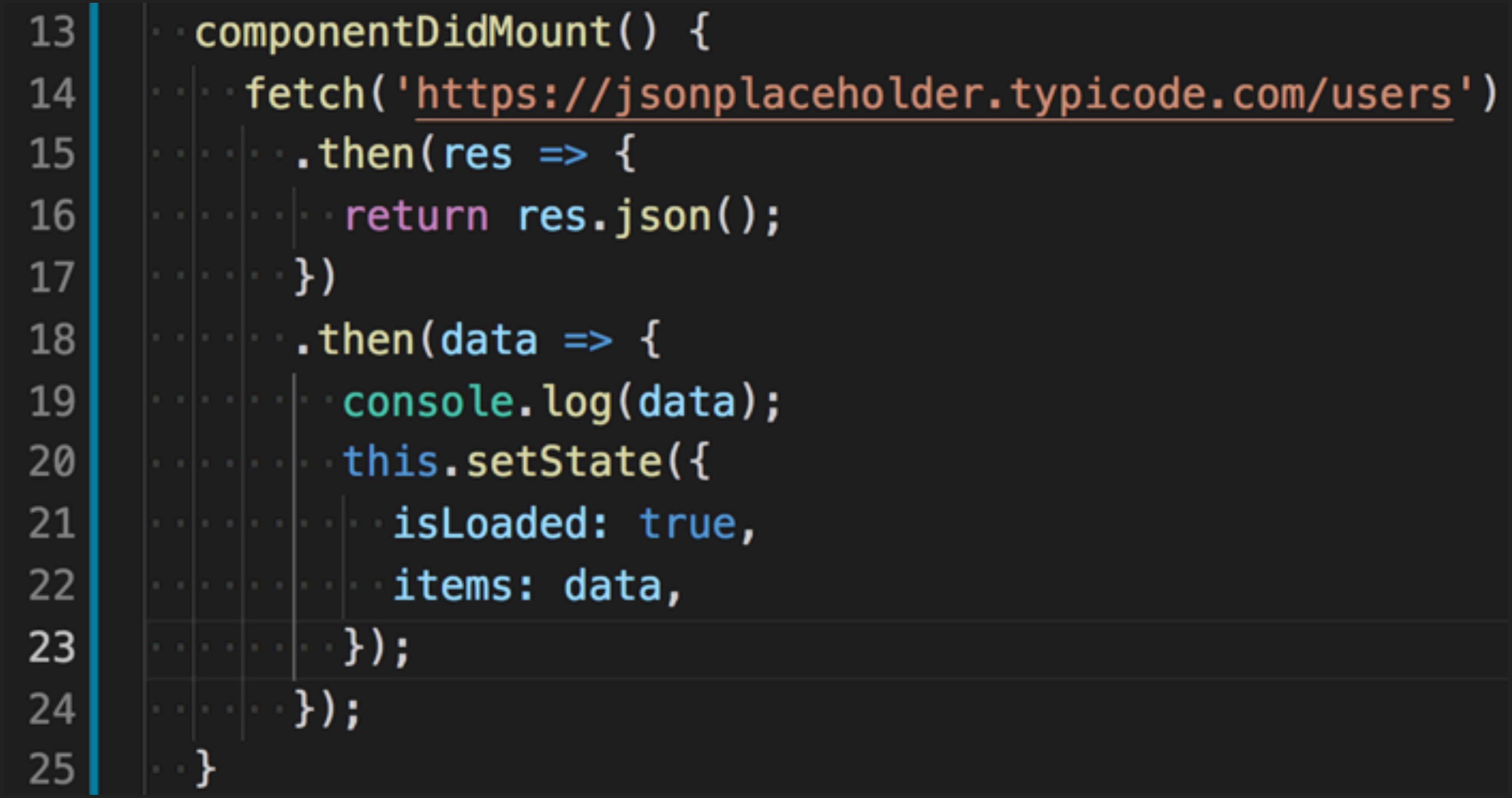


```
  componentDidMount() {
    //put code here
  }
```

Remember to put methods after the constructor function and before your render method

This is a built in React method, these built-in lifecycle methods follow this format

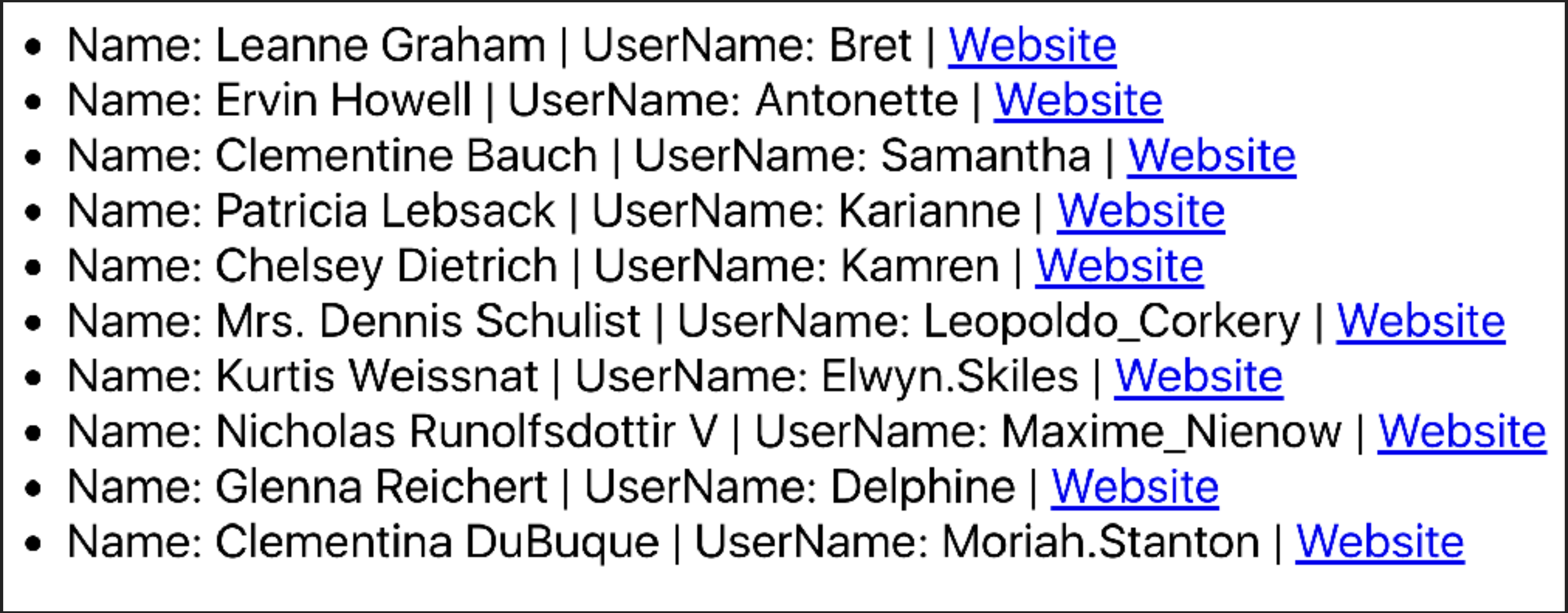
- Now that we have our lifecycle method setup. We want to accomplish this API request by using a built-in method called Fetch. This method will allow us to use a specific URL to grab the data from, and then process the data by using the .then methods.



```
13    componentDidMount() {
14      fetch('https://jsonplaceholder.typicode.com/users')
15        .then(res => {
16          return res.json();
17        })
18        .then(data => {
19          console.log(data);
20          this.setState({
21            isLoaded: true,
22            items: data,
23          });
24        });
25    }
```

The console.log on line 19 is for you to look at the data that is coming in from the response. Be sure that this data is formatted in a way you can access the indices or key value pairs.

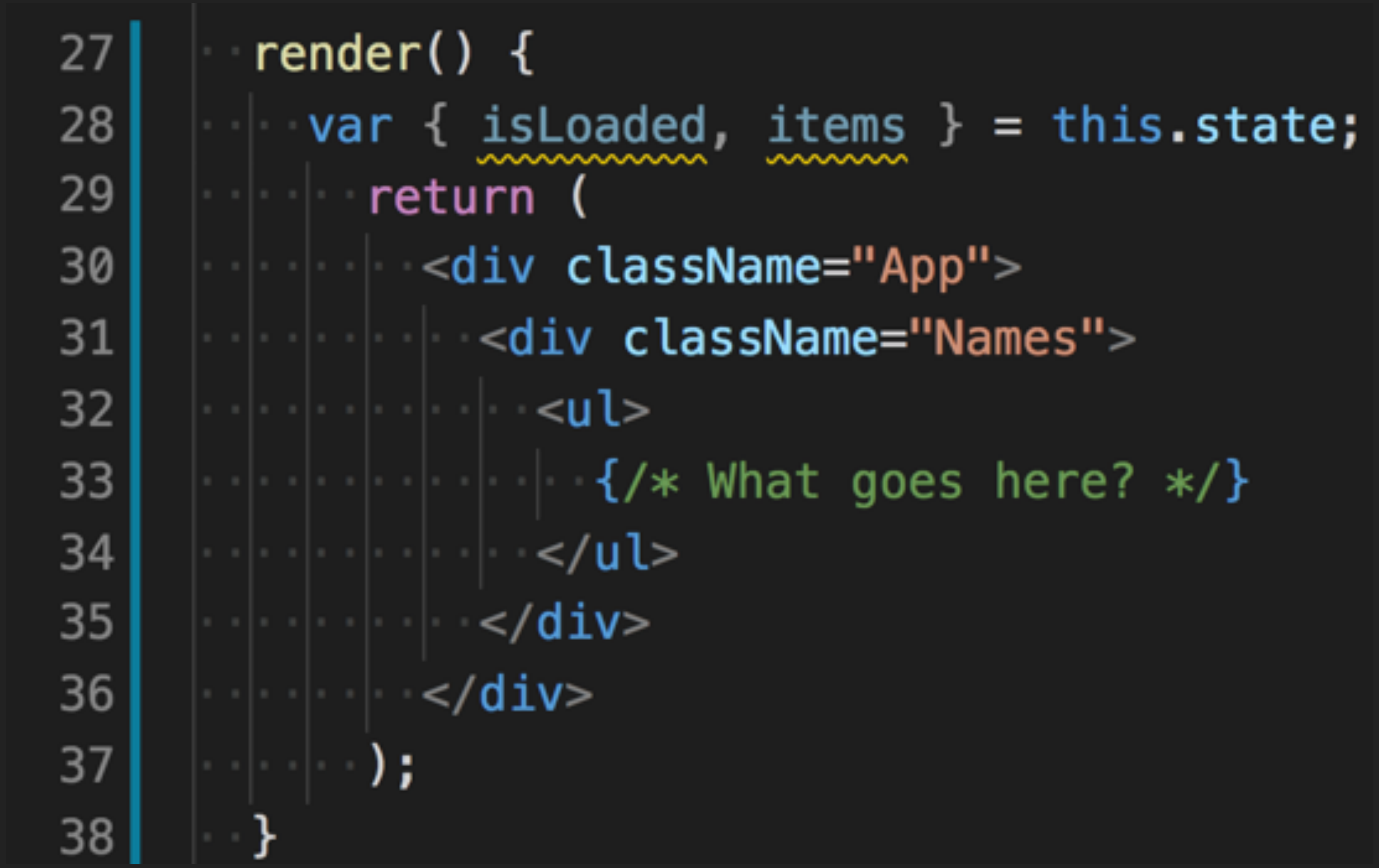
- Now that we've fetched some data, and added this data to state, as well as setting the isLoaded value to true. We now need to look at our expected end result and build out our JSX.



- Name: Leanne Graham | UserName: Bret | Website
- Name: Ervin Howell | UserName: Antonette | Website
- Name: Clementine Bauch | UserName: Samantha | Website
- Name: Patricia Lebsack | UserName: Karianne | Website
- Name: Chelsey Dietrich | UserName: Kamren | Website
- Name: Mrs. Dennis Schulist | UserName: Leopoldo_Corkery | Website
- Name: Kurtis Weissnat | UserName: Elwyn.Skiles | Website
- Name: Nicholas Runolfsdottir V | UserName: Maxime_Nienow | Website
- Name: Glenna Reichert | UserName: Delphine | Website
- Name: Clementina DuBuque | UserName: Moriah.Stanton | Website

LIFECYCLE WORKSHOP

- First lets deconstruct our state keys by typing line 28 below the starting code block of the render method and before the return statement.



```
27   render() {
28     var { isLoaded, items } = this.state;
29       return (
30         <div className="App">
31           <div className="Names">
32             <ul>
33               {/* What goes here? */}
34             </ul>
35           </div>
36         </div>
37       );
38   }
```

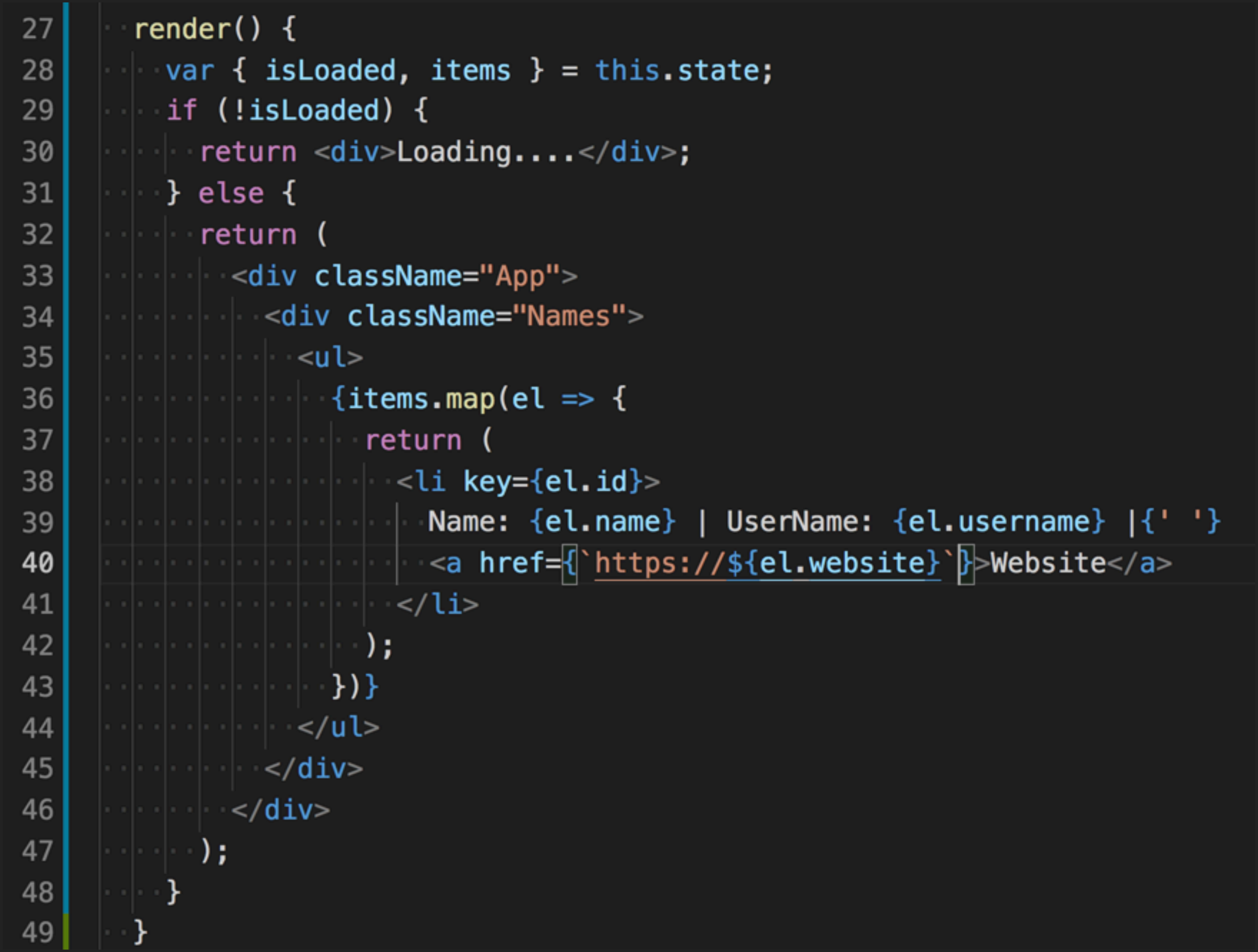
Think about what specific values are supposed to be displayed and how to accomplish this by a javascript array method.

Conditional Rendering

- Take a moment and think about the React lifecycle. Since we used the componentDidMount() lifecycle method. The component has already rendered to the DOM. Now we can create a state change by using a lifecycle method, and remember that once we do a state change, the render method re-renders and attaches the changes to the DOM.
- This is where conditional rendering comes in. When you're doing network calls, you want to handle the case when you have a lot of data coming in that takes more than a microsecond to gather.

- This is where conditional rendering come in. You've seen it before where you wait for the loading screen to disappear, and then show you the content you requested. Now what we need to do is use our isLoaded state value that changes to true once the data is completely fetched and use that to render the list when its completely populated.

LIFECYCLE WORKSHOP



```
  render() {
    var { isLoaded, items } = this.state;
    if (!isLoaded) {
      return <div>Loading....</div>;
    } else {
      return (
        <div className="App">
          <div className="Names">
            <ul>
              {items.map(el => {
                return (
                  <li key={el.id}>
                    Name: {el.name} | UserName: {el.username} |{' '}
                    <a href={`https://${el.website}`}>Website</a>
                  </li>
                );
              })}
            </ul>
          </div>
        </div>
      );
    }
  }
```

- Conditional rendering
- Fetch
- Array.map()
- Lifecycle methods
- String interpolation