The generated files, *index.js* and *routes.js*, are similar to the previously created ones. TSC doesn't touch the static HTML. The stand-alone Babel.js script converts the JSX code on runtime in the browser. Start the server from your command line:

```
$ node index.js
Listening on 3000
```

Now visit *http://localhost:3000/components/weather-component* in your browser. You see the same text you saw when you rendered the weather component in the React.js playground, as in Figure 4-2. As soon as you click the text, the click handler increases the reactive state variable, and the counter shows the new value.

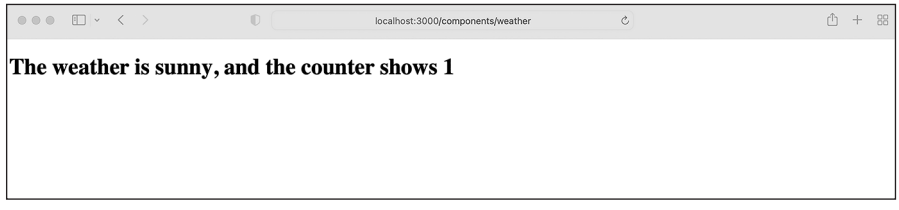


*Figure 4-2: The response our browser receives from our Node.js web server*

You successfully created your first React.js application. To gain more experience with React.js, try adding a custom button component for the click counter, with a style attribute that uses a JSX expression to change the background color for odd and even counter values.

## Summary

You should now have a solid foundation with which to create your React.js apps. JSX elements are the building blocks of React.js components that return JSX to be rendered as HTML in the DOM, via React.js's virtual DOM. You also explored the difference between class components and modern function components, took a deep dive into React.js hooks, and used these hooks to build a functional component.

If you want to explore React.js's full potential, take a look at the React.js tutorials from W3Schools at *https://www.w3schools.com/REACT/DEFAULT.ASP* and those created by the React.js team at *https://reactjs.org/tutorial/tutorial.html*.

In the next chapter, we'll work with Next.js. Built on top of React.js, Next.js is a production-ready full-stack web development framework for single-page applications.