Recap: State in React

Overview

React state is the cornerstone of making React apps interactive – the same applies to your table booking app. To prepare for adding state to your project, you'll brush up on your knowledge of working with state in React in this reading.

React state overview

State is an object. You can control it using the `useState` hook. For example, consider the following code of a task list in a to-do app:

```
1   import React, { useState } from "react";
2   import { TaskList } from "./components/TaskList";
3
4   export default function App() {
5     const [tasks, setTasks] = useState([
6       { id: 1, task: "Go shopping", done: true },
7       { id: 2, task: "Wash dishes", done: false },
8     ]);
9
10    return (
11      <TaskList tasks={tasks} />
12    )
13  }
14
```

There are several important concepts to revisit in the above code example, namely:

- Object destructuring on line 1
- Array destructuring of `useState` variables on line 5
- Using an array of objects for the initial state value on lines 5 to 8
- Passing in state data as props from the parent component to a child component

Object destructuring

Object destructuring is a common pattern in React. Inspect the first line of code again:

```
1   import React, { useState } from "react";
```

In this line, React is imported and then `{ useState }` is imported. This allows you to write `useState` instead of `React.useState` in the rest of the App component's code, without accessing it through the React object. This pattern is commonly used for imports.

Array destructuring of `useState` variables

The code that begins on line five of the code example provided earlier is using the `useState` hook. Here's a slight variation on the given code, so that it's easier to describe what is happening:

```
const [tasks, setTasks] = useState()
```

As mentioned already, this is an example of array destructuring. The return value of a `useState` call is always an array.

Tip: The advantage of array destructuring over object destructuring is that you can destructure an object using only the exact property key of the source object, while you can destructure a member from an array using any name.

Using an array of objects for the initial state value

The initial state value of the task list app example is using an array of objects. This is because it is a very convenient way to work with data in JavaScript. In fact, it is so convenient that many APIs return arrays of objects when their data is fetched.

Back to the example, what will the value of the tasks variable in the example code below be?

```
import React, { useState } from "react";
import { TaskList } from "./components/TaskList";

export default function App() {
  const [tasks, setTasks] = useState([
    { id: 1, task: "Go shopping", done: true },
    { id: 2, task: "Wash dishes", done: false },
  ]);

  return (
    <TaskList tasks={tasks} />
  )
}

```

The starting value of the tasks variable will be:

```
[
    {id: 1, task: "Go shopping", done: true},
    {id: 2, task: "Wash dishes", done: false }
]
```

Having this data structure as the starting code of the task list allows you to easily create, read, update and delete the tasks array's members, that is, the individual tasks.

Passing in state data as props from the parent component to a child component

Take note of the return value of the App component:

```
return (
    <TaskList tasks={tasks} />
)
```

This is the rendering mechanism of React. It uses the array of task objects and it passes it to the `TaskList` component. Then, if you were actually building this app, you'd code the `TaskList` component so that it accepts the `props.tasks` and does something with the data that comes in.

Conclusion

In this reading, you revisited the concept of state in a react app. Now that you've recapped state and its related concepts, it's time to complete your next exercise, Adding Table-Booking State. While this recap has given you a brief overview of working with state in React, if you would like a more in-depth refresher on these topics, you can revisit the following items from the **React basics** course:

- What is state
- Observing state
- Managing state
- Data Flow in React

Additionally, you might want to check out the following lesson items from the **Advanced React** course:

- What you know about props and state
- Revising useState hook
- Working with complex data in useState
- Using the useState hook