COURSEWARE

State

Contents

Overview

In this module, we will be looking at states.

The State

Data that can change should be considered as state.

State should be the single source of truth for changing data.

All components that rely on this should receive the data as props.

State should be in the highest common component of those that require the data.

Identifying the Minimal UI

React suggests we identify the minimal (complete) representation of UI state.

This is made easy through use of state; it's best practice to first think of the minimal set of mutable states the app needs.

Everything else can be computed on demand, such as a list of products, the search text user enters, the value of checkbox, a filtered list of products, etc.

To figure out state, ask:

- Is it passed in from a parent via props? If so, it probably isn't state
- Does it remain unchanged over time? If so, it probably isn't state
- Can you compute it based on any other state or props in the component? If so, it probably isn't state

So for the above examples, which would be a state?

- Original list of products would be passed in as props – **not state**
- Search text user enters – **state**
- Value of checkbox – **state**
- Filtered list of products – can be computed by combining original list of object, search text and checkbox value – **not state**

State location

Identifying where the state lives involves identifying which component mutates or owns the state

React focuses on a one-way data flow-down component hierarchy, so whilst it may not be immediately clear which component should own state, it can be identified.

To work out where state should live:

- Identify every component that renders something based on state
- Find common owner component
- Either common component or component even higher up should own state
- If no component makes sense, create new component to hold state and add it into the hierarchy above the common owner component

For the example:

- `ProductTable` needs to filter the product list based on state, and `SearchBar` needs to display search text and checked state
- Common owner component will be `FilterableProductTable`
- Conceptually makes sense for filter text and checked value to live in FilterableProductTable

State and Functions

Previous to React v16.8, state was not allowed in Function components.

Hooks were introduced to allows Function components to have state.

`{ useState }` needs to be imported from React to allow this functionality.

Declaring state needs the destructured setting of an array containing the state name and a function to update it assigned to a call to useState with an initial value.

On the initial render, the initial value passed will be used, but this can then be adjusted!

```
// Abstract const
const [myState, setMyState] = useState(initialMyState);

// Component example
import { useState } from 'react';

const App = () => {
    const [count, setCount] = useState(0);
    // count initially set to 0

    //after every button click, add 1 to the count
    return(
        <>
            <p>Count is: {count}</p>
            <button onClick={() =>setCount(count + 1)}>Add</button>
        </>
    );
};
export default App;
// Would display: Count is: 0
// After 3 clicks, would display: Count is: 3
```

Tutorial

Adding states using hooks

Let's create the above example; a page that displays the number of steps we have taken today.

1. Create a jsx file called `MilesAhead.jsx`
2. Import `useState` from `React`

   ```
   import {useState} from 'react';
   ```

3. Create a const called `MilesAhead` as an arrow function that takes no arguments

```
const MilesAhead = () => {}
```

4. Use the 'useState' to set the value of `[steps, setSteps]` to have an initial value of 0.

```
const [steps, setSteps] = useState(0);
```

5. Declare a function called increment which increases the value of `step`

```
const increment = () => {
    setSteps(prev => prev + 1);
}
```

6. Write a return statement which prints the number of steps taken today and includes a button which calls the `increment()` function onclick.

```
return(
    <>
        <p>Today you've taken {steps} steps!</p>
        <br/>
        <button onClick={increment}>I took another step </button>
    </>
);
```

7. Export the component and call it in the return of `App.js`.
8. Run the program using `npm start`

▶ MilesAhead.jsx

▶ Adding states to a class

Exercises

1. Create a component that saves customer's username and password from the input of a form - use hooks for this exercise.

▶ Solution

2. Create a car component which stores the below properties in the state - render a page which retrieves the values from the state and prints them to the screen.
    1. Brand
    2. Model
    3. Colour
    4. Year
3. Write code so that the state of each of the properties can be modified by user input.

▶ Solution

4. Create a `ProductTable` component that filters information based on the vaule of a Search field.