React State

Learn how React state is used to make applications interactive.

We'll cover the following

- Exercises:

React Props are used to pass information down the component tree; **React state** is used to make applications interactive. We'll be able to change the application's appearance by interacting with it.

First, there is a utility function called `useState` that we take from `React` for managing state. The `useState` function is called a hook. There is more than one **React hook** – related to state management but also other things in React – and you will learn about them throughout the next sections. For now, let's focus on **React's `useState` hook**:

```
const App = () => {
  const stories = [ ... ];

  const [searchTerm, setSearchTerm] = React.useState('');

  ...
};
```

src/App.js

React's `useState` hook takes an *initial state* as an argument. We'll use an empty string, and the function will return an array with two values. The first value (`searchTerm`) represents the *current state*; the second value is a *function to update this state* (`setSearchTerm`). I will sometimes refer to this function as *state updater function*.

If you are not familiar with the syntax of the two values from the returned array, consider reading about JavaScript array destructuring. It is used to read from an array more concisely. This is array destructuring and its benefits visualized in a nutshell:

```
// basic array definition
const list = ['a', 'b'];


// no array destructuring
const itemOne = list[0];
const itemTwo = list[1];

// array destructuring
const [firstItem, secondItem] = list;
```

src/App.js

In the case of React, the React `useState` hook is a function which returns an array. Take again the following JavaScript example as comparison:

```
function getAlphabet() {
  return ['a', 'b'];
}

// no array destructuring
const itemOne = getAlphabet()[0];
const itemTwo = getAlphabet()[1];

// array destructuring
const [firstItem, secondItem] = getAlphabet();
```

src/App.js

Array destructuring is just a shorthand version of accessing each item one by one. If you express it without the array destructuring in React, it becomes less readable:

```
const App = () => {
  const stories = [ ... ];

  // less readable version without array destructuring
  const searchTermState = React.useState('');
  const searchTerm = searchTermState[0];
  const setSearchTerm = searchTermState[1];

  ...
};
```

src/App.js

The React team chose array destructuring because of its concise syntax and ability to name destructured variables. The following code snippet is an example of array destructuring:

```
const App = () => {
  const stories = [ ... ];
```

```
  const [searchTerm, setSearchTerm] = React.useState('');


  ...
};
```

src/App.js

After we initialize the state and have access to the current state and the state updater function, use them to display the current state and update it within the App component's event handler:

```
const App = () => {
  const stories = [ ... ];

  const [searchTerm, setSearchTerm] = React.useState('');

  const handleChange = event => {

    setSearchTerm(event.target.value);

  };

  return (
    <div>
      <h1>My Hacker Stories</h1>

      <label htmlFor="search">Search: </label>
      <input id="search" type="text" onChange={handleChange} />

      <p>
        Searching for <strong>{searchTerm}</strong>.
      </p>
      <hr />

      <List list={stories} />
    </div>
  );
};
```

src/App.js

The following code is the complete demonstration of the above concepts:


```
  □ □ □□  □   ã□  F   □□  □   □  )□      □   9□  5□  @@  □   °□  n□  □PNG
□
   IHDR   □   □□□   (-□S   äPLTE""""""""""""""""""2PX=r□)7;*:>H□¤-BGE□□8do5Xb6[eK□®K□¯1MU9gs3S
□
   IHDR   □   □□□   ×©ÍÊ   □ePLTE""""""""""""""""""""""""2RZN¢¹J□«3R[J□¬)59YÁÞ0KS4W`Q«ÄL□²%+-0JR
?^q÷ñíÛ□i.},□isæÝ_TttÔ¾ □1#□□/(ì□-[□□□è`□è`Ì□ÚïÅðZ□d5□□□□?ÎebZ¿Þ□i.Ûæ□□□ìqÎ□+1°□}Â□5ù  ïçd
□
   IHDR        □□   D¤□Æ   □APLTE   """"""""""""""""""""""""""2RZVºÖ_ÔôU·Ñ=r□$()'25]ÎíC□□0LS<o}X
□
   IHDR   @   @□□   □·□ì   □:PLTE   """"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
¢ßqÇ8Ù□´□mKË±mÆ¶mÛü·yi!è□ÎªYÏuë ÀÏ_Àï?i÷□ý+ò□□ÄA□|□ù{□□´?¿□_En□).□JËD¤<□
©¬¢Z\Ts©R*□(□  ¯©□J□□□□u□X/□4J□9□¡5·DEµ4kÇ4□&i¥V4Ú□¡®Ð□□¯□vsf:àg,□¢èBC»î$¶□ºÍùî□□á□@□ô□I
```

-ê>Û□º«¢XÕ¢î}ß¨ëÛÑ;□ÃöN´□ØvÅý□Î¸ÿ1 □ë×ÄO@&v/Äþ_□ö\ô□Ç\í.□□½+0□□;□□□!□fÊ□¦´Ó%Â JY·O□Â□'/Å]_□

When the user types into the input field, the input field's change event is captured by the handler with its current internal value. The handler's logic uses the state updater function to set the new state. After the new state is set in a component, the component renders again, meaning the component function runs again. The new state becomes the current state and can be displayed in the component's JSX.

Exercises:

- Confirm the changes from the last section.
- Read more about JavaScript array destructuring.
- Read more about React's useState Hook ([0], [1]), as it makes your React components interactive.