REACT HOOKS

YT Course by The Net Ninja / June 2019

Mind map by Ana-Maria Dobre

OVERVIEW

Hooks:
- special fct
- allow us to do additional things inside FC (typically only possible within CC, for example, things like using state)

Examples of hooks:
useState()
// use state within a FC

useEffect()
// run code when a component renders (or re-renders)

useContext()
// consume context in a FC

Getting started:
Setup a project and arrange the stage for showcasing our first hook.
Create new React application (1)

Create stateless FC called SongList, like so (2)
// returns very simple template

Import new component into App.js like so (3)

(1)
C:\Workspace\nn\contextapp>**npx create-react-app hooksapp**

(2) // SongList.js
```
import React from "react";

const SongList = () => {
  return(
   <div className="song-list">
    <ul>
     <li>Rosas</li>
     <li>Lo que te conté mientras te hacias la dormida</li>
     <li>La paz de tus ojos</li>
    </ul>
   </div>
  );
}

export default SongList;
```

(3) // App.js
```
import SongList from "./components/SongList";

function App() {
 return (
  <div className="App">
   <SongList />
  </div>
 );
}

export default App;
```

useState HOOK

How to use useState:
- invoke **useState** fct inside SongList FC
// allows you to use a piece of state inside this FC => import {useState}
- accepts an argument representing initial value
// in our case, it will be an array of songs
- the useState fct returns an array of two values:
1) the piece of state itself
2) a fct that we can use to edit that piece of state
=> use array destructuring to get these 2 different values.

Suggestion: call the first property **songs** (because that is what the state defines); call the second property **"set" (fixed) + Songs** (name of the first property)

=> we can now cycle through the songs array inside the template, using **map** (no need to hardcode each song)
// **songs.map(song => { ... })**
// react expects us to provide a key prop, unique for each li element

Additionally, may add a button in order to add a new song.
Steps required:
- attach an **onClick event handler** to the button, referencing a fct called addSong // arrow fct
- inside the addSong fct, change the data by using setSongs
- call setSongs and pass it a new value that will completely replace existing value => need to use spread syntax (get the current songs, *spread* them into new array, which will additionally contain a new song object)

Test:
Click "**Add a song**" button several times. You would get a warning (2)

Fix previous warning:
- install uuid library (3)
- import it in the SongList component as such (4)
- use it (5) // generates a unique id every time

Remember:
useState returns an array with 2 values:
- the actual value of the state
- a fct to change/edit that value

(1) // SongList.js
```
import React, {useState} from "react";

const SongList = () => {
   const [songs, setSongs] = useState([
     {title: 'Lo que te conté mientras te hacias la dormida', id: 1},
     {title: 'Rosas', id: 2},
     {title: 'Las paz de tus ojos', id: 3}
   ]);

   const addSong=() => {
     setSongs([...songs, {title: 'Tu recuerdo', id: 4}])
   }

   return(
     <div className="song-list">
       <ul>
         {songs.map(song => {
           return(
             <li key={song.id}>{song.title}</li>
           );
         })}
       </ul>
       <button onClick={addSong}>Add a song</button>
     </div>
   );
}

export default SongList;
```

(2)
Warning: Encountered two children with the same key, `4`. Keys should be unique so that components maintain their identity across updates. Non-unique keys may cause children to be duplicated and/or omitted — the behavior is unsupported and could change in a future version.

(3)
npm install uuid

(4)
import {v4 as uuidv4} from 'uuid'

(5)
setSongs([...songs, {title: 'Tu recuerdo', id: uuidv4()}])

useState WITH FORMS

Idea: instead of the button in component SongList, add a form.
=> Create new component **NewSongForm.js** (1)
// **initially stateless**, but we will add state later

Go to SongList.js, remove button.
=> replace with <NewSongForm /> tag // it generates auto-import

Review form to make some changes, such as: (2)

- **attach event listener to the form** // <form **onSubmit** = {}>
- **track what user types into the input field**
// input type="text" **onChange**={(e) => { ... })
// whenever the value changes, after each keystroke, a fct is fired; inside this fct, store what the user types in, **into some kind of local state. The fct takes an event object.**
- to use that local state, use **hook useState** // import it
- passing in an initial value into useState // **useState('');**
- update the state, on every change, after each keystroke, with whatever the user has typed in

Reminder:
We get an array from calling useState, with 2 values:
1) actual state 2) fct to update that state
=> const [title, setTitle] = useState('');

- use **setTitle** to update the state with whatever user is typing in
// input type="text" onChange={(e) => { setState(e.target.value) })
=> keeping title in sync with the value in the input field
- attach (on the input field) a **value** prop and set it equal to title
// input type="text" value={title} onChange={(e) => setState(e.target.value))
- create fct to handle the submit // onSubmit = {handleSubmit}
- define **handleSubmit** fct on the form
Set it equal to an array fct, take an event object, and do the following: first, prevent default form action when submitting form i.e. refresh page, then log the title

(1) // NewSongForm.js
```
import React from "react";

const NewSongForm = () => {
  return (
    <form>
      <label>Song name:</label>
      <input type="text" required />
      <input type="submit" value="add song" />
    </form>
  )
}

export default NewSongForm;
```

(2) // NewSongForm.js (enhanced to log inputted value on every form submit)
```
import React, {useState} from "react";

const NewSongForm = (() => {
  const [title, setTitle] = useState('');
  const handleSubmit = (e) => {
    e.preventDefault();
    // console.log(title);
    setTitle('');
  }
  return (
    <form onSubmit={handleSubmit}>
      <label>Song name:</label>
      <input type="text" value={title} required onChange={(e) => setTitle(e.target.value)}/>
      <input type="submit" value="add song" />
    </form>
  )
}

export default NewSongForm;
```

(3) // SongList.js
```
import React, {useState} from "react";
import {v4 as uuidv4} from 'uuid'
import NewSongForm from "./NewSongForm";

const SongList = () => {
  const [songs, setSongs] = useState([
    {title: 'Lo que te conté mientras te hacias la dormida', id: 1},
    {title: 'Rosas', id: 2},
    {title: 'Las paz de tus ojos', id: 3}
  ]);

  const addSong=(title) => {
    setSongs([...songs, {title: title, id: uuidv4()}])
  }

  return(
    <div className="song-list">
      <ul>
        {songs.map(song => {
          return(
            <li key={song.id}>{song.title}</li>
          );
        })}
      </ul>
      <NewSongForm addSong={addSong} />
    </div>
  );
}

export default SongList;
```

Next: need to send the song that you just logged to the console from NewSongForm component (child)
to SongList component (parent)

Go to **SongList.js** > slightly modify method **addSong** by adding an argument i.e. title (3)
// inside addSong, we are calling setTitle passing title (where we used to have a hardcoded value)

Still in SongList.js, **pass the addSong fct as a prop to the child (NewSongForm)**. // <NewSongForm addSong={addSong} />

Back into NewSongForm.js (4), **accept the prop inside an object**, so // const NewSongForm = ({addSong}) => { ...
and instead of logging the title, call addSong(title)

Notes

FC: functional components
CC: class components