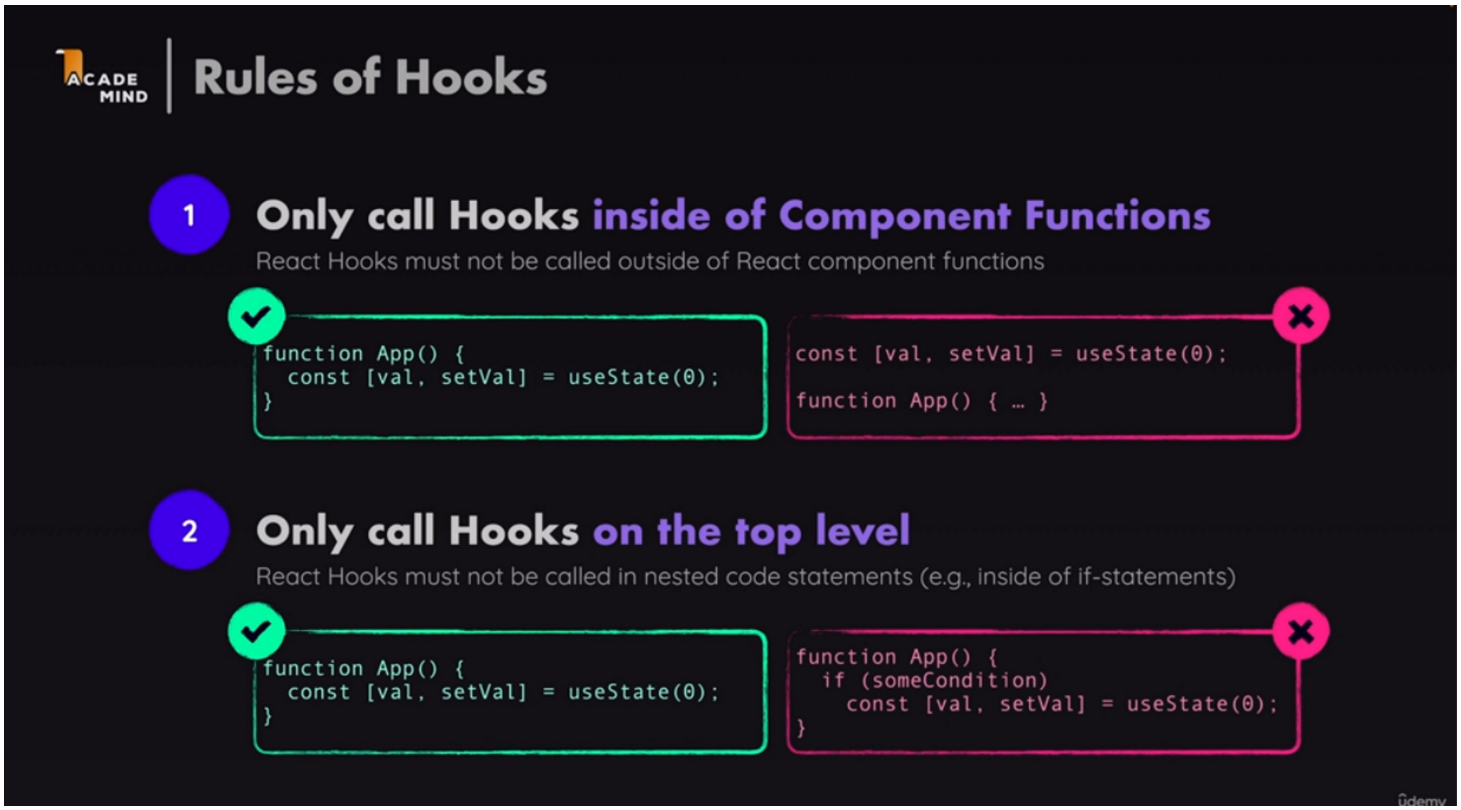
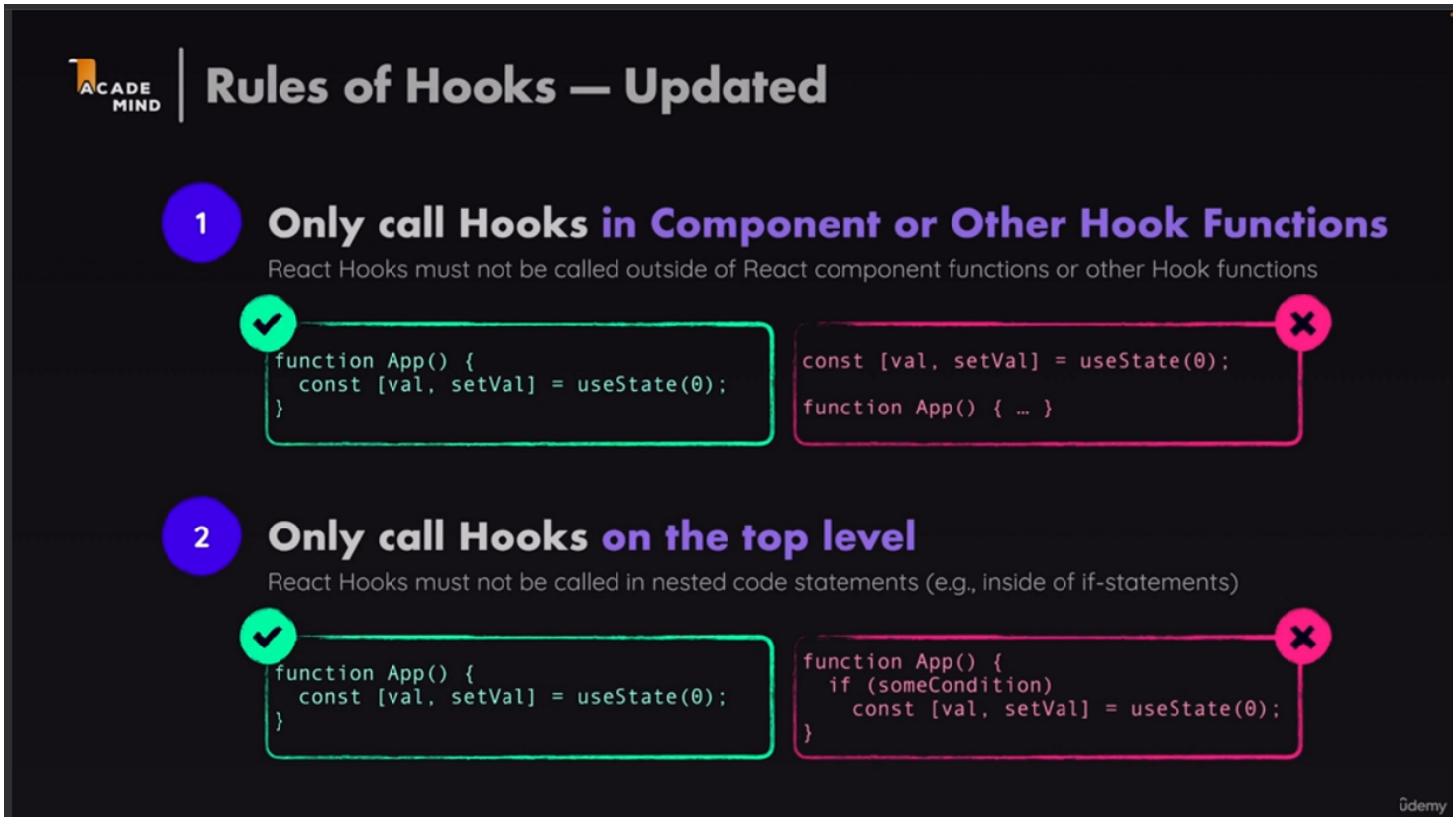
Sec 16 - Building Custom React Hooks

Sunday, September 29, 2024 9:40 PM





The idea behind building custom hooks is always to wrap and reuse code that goes into your Component functions.

So Hooks may be used inside of Component functions or inside of other Hooks. That's an important variation of this first rule
and it's this variation that enables us to build custom Hooks, as you will see throughout this section.

But why would you build custom Hooks and what should these custom Hooks do? Well, the idea behind building custom Hooks is always to wrap and reuse code
that goes into your Component functions.

For example, here in the App.jsx file in this project I provided to you. There, I got this useEffect Hook where I in the end send an HTTP request
and where I manage some loading state and potentially an error state.

```jsx
function App() {
  const selectedPlace = useRef();

  const [userPlaces, setUserPlaces] = useState([]);
  const [isFetching, setIsFetching] = useState(false);
  const [error, setError] = useState();

  const [errorUpdatingPlaces, setErrorUpdatingPlaces] = useState();

  const [modalIsOpen, setModalIsOpen] = useState(false);

  useEffect(() => {
    async function fetchPlaces() {
      setIsFetching(true);
      try {
        const places = await fetchUserPlaces();
        setUserPlaces(places);
      } catch (error) {
        setError({ message: error.message || 'Failed to fetch user places.' });
      }

      setIsFetching(false);
    }

    fetchPlaces();
  }, []);
```

and you then take a look at the code in the AvailablePlaces.jsx file, you will notice that the code is quite similar.

```jsx
export default function AvailablePlaces({ onSelectPlace }) {
  const [error, setError] = useState();

  useEffect(() => {
    async function fetchPlaces() {
      setIsFetching(true);

      try {
        const places = await fetchAvailablePlaces();

        navigator.geolocation.getCurrentPosition((position) => {
          const sortedPlaces = sortPlacesByDistance(
            places,
            position.coords.latitude,
            position.coords.longitude
          );
          setAvailablePlaces(sortedPlaces);
          setIsFetching(false);
        });
      } catch (error) {
        setError({
          message:
            error.message || 'Could not fetch places, please try again later.',
        });
        setIsFetching(false);
      }
    }
```

It's not exactly equal, we are sending different requests to different URLs and here in AvailablePlaces we are then also transforming the data we're getting back, but the main idea is the same.

We're sending an HTTP request, we're managing some loading state, we're managing some error state and we're using useEffect for all that. And this would be an example for some code that we might want to outsource and make more generic so that we can reuse it,
so that we don't have to write this similar code twice in two different Components, but so that we maybe have one reusable configurable function
that could be used in both Components.

Because, of course, you might remember that the idea behind Components was to reuse Logic and JSX structure
as we're doing it here with the modal, for example, we are reusing this Component with different data.

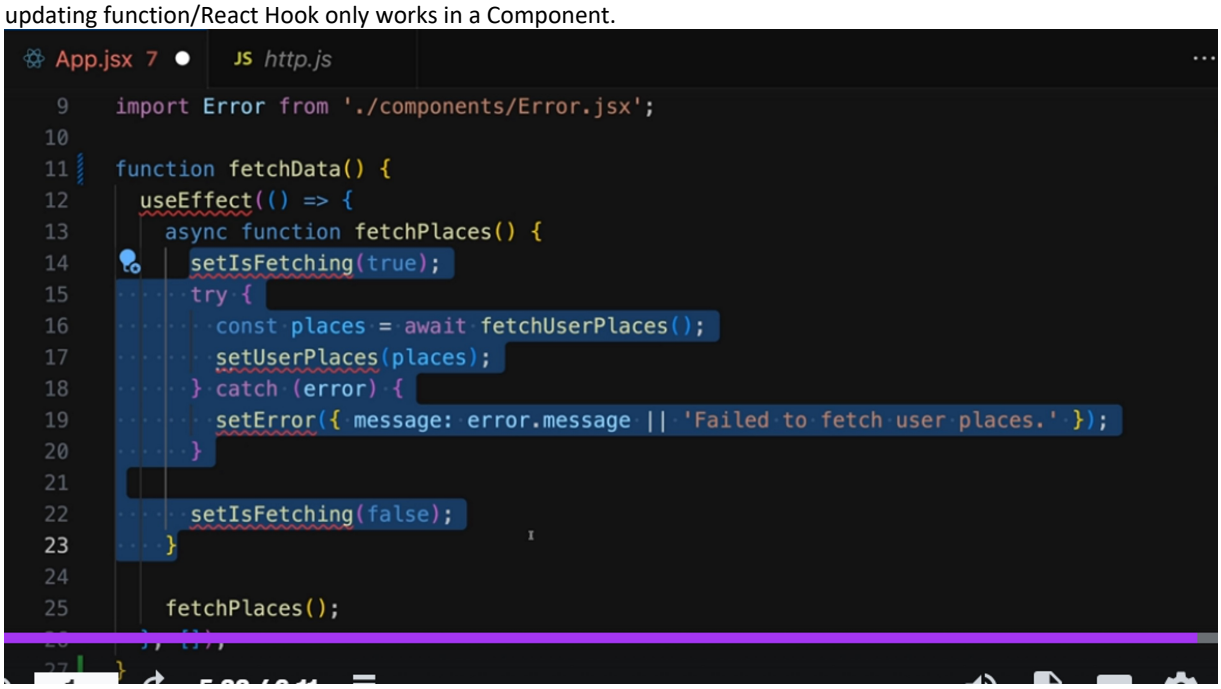
And just as this was the idea behind Components we might want to reuse other code as well, code that does not return JSX code so that can't go into its own Component, for example. Code like this, this useEffect call with the logic inside of it, we might want to reuse that,
but we can't put it into a separate Component because this is just a part of a Component.

It does not return JSX code or anything like that and therefore this is the kind of problem we can solve with custom React Hooks.
Now of course, we could also try t outsource and reuse code like this by grabbing it and putting it into a separate function, a function which we could call fetchData and which we could define in any file we want.

And then we could try to call fetchData in every Component that needs that functionality because that is how you normally outsource
logic in programming and JavaScript programming is, of course, no exception.

If you have some logic that needs to be used in different places, you put it into a function and that function can then be used in different places.

You might think I can write a separate function and outsource it? But as we already know, state updating function/React Hook only works in a Component.



And that's why this will not be a valid solution and why I'll roll back this code and go back to the code I had before.

```
App.jsx    http.js
ooks" & Why To Use Hooks
20    const [modalIsOpen, setModalIsOpen] = useState(false);
21
22    useEffect(() => {
23      async function fetchPlaces() {
24        setIsFetching(true);
25        try {
26          const places = await fetchUserPlaces();
27          setUserPlaces(places);
28        } catch (error) {
29          setError({ message: error.message || 'Failed to fetch user places.' });
30        }
31
32        setIsFetching(false);
33      }
34
35      fetchPlaces();
36    }, []);
```

And that's exactly why we need this custom Hooks featureI'll introduce you to in this section because custom Hooks will also be functions that we can then call from different places, but they will be functions that are guaranteed to be used in valid places, so that are guaranteed to be used in Components in the end.

Functions that start with 'use' are treated as Hooks.

and React projects typically look for functions that start
with use and enforce certain rules on such functions.

For example, in App.jsx, if I would try to use the built-in useState hook nested inside of some other code, you would see that here I'm getting an error. I'm getting red squiggly lines

```
App.jsx 9+ ●    JS useFetch.js 2, U ●
16    const [error, setError] = useState();
17
18    const [errorUpdatingPlaces, setErrorUpdatingPlaces] = useState();
19
20    const [modalIsOpen, setModalIsOpen] = useState(false);
21
22    useEffect(() => {
23      const [userPlaces, setUserPlaces] = useState([]);
24
25      React Hook "useState" cannot be called inside a callback. React
26      Hooks must be called in a React function component or a custom
27      React Hook function. eslint(react-hooks/rules-of-hooks)
28      (alias) useState<never[]>(initialState: never[] | (() =>
29      never[])): [never[],
30      React.Dispatch<Re[]>>] (+1 overload)
```

I'm getting red squiggly lines

If the state gets updated in the custom hook, it will also trigger a re-render of the component in which you are using the custom hook.

For Example, here when we

setIsFetching(true);

```
const [isFetching, setIsFetching] = useState();
const [error, setError] = useState();
const [fetchedData, setFetchedData] = useState();

useEffect(() => {
  async function fetchData() {
    setIsFetching(true);
    try {
      const data = await fetchFn();
      setFetchedData(data);
    } catch (error) {
      setError({ message: error.message || 'Failed to fetch data.' });
    }

    setIsFetching(false);
  }

  fetchData();
```

When that happens, the component where your custom hook gets used also gets executed again. So that still works as if you would be managing the state directly inside of the component. That still works exactly the same way even when using a custom hook as an extra layer.

So, the behavior of this App Component will not change.

```
App.jsx 9+ ●    JS useFetch.js U ●

import { fetchUserPlaces, updateUserPlaces } from './http.js';
import Error from './components/Error.jsx';
import { useFetch } from './hooks/useFetch.js';

function App() {
  const selectedPlace = useRef();

  const [errorUpdatingPlaces, setErrorUpdatingPlaces] = useState();

  const [modalIsOpen, setModalIsOpen] = useState(false);

  const { isFetching, error, fetchedData } = useFetch(fetchUserPlaces);

  function handleStartRemovePlace(place) {
    setModalIsOpen(true);
    selectedPlace.current = place;
  }
```