Go to ToC

React Hooks

ToC

Load Data From Web API

- We've had a situation where our data was statically set inside of our component

```
const user = [
    // Data was here
]

return (
    <Profiles users={users}>
)
```

- Let's write a module where it helps us to retrieve data from an API instead of statically stored data.

```
// users.js file
function load() {
    const data = fetch('https://reqres.in/api')
    .then(response => {
        return response.json();
    })
    .then(results => results.data)
    .catch(err) {
        console.log(err);
    }

    return data;
}

export default load;
```

TOP

`useEffect` Hook

- Now we have to use it in our `App` component, but we can't really do the following:

```
const users = load();
```

- Since we're working with React and `load` is a promise, React takes care of **when** things are happening. So we have to use "hooks" which will manage when to do side effects.
- So we have to use `useEffect` hook from 'react' library

```
import {useEffect} from 'react'

function App() {

    useEffect( () => {
        load().then(data => console.log(data))
    })

    return (
        <div className='App'>
            <Profiles users={users}>
        </div>
    )
}
```

TOP

`UseState` Hook

- Now this code works, but there's no way to store data that is coming from an API call. We could have used object instance as a state in class components, but in functional components , we have to have a way to store state somewhere outside of the function. This is where we can take advantage of `useState` hook.

```
import {useEffect, useState} from 'react'

function App() {
    // users - an array
    // setUsers - function to update `users`
    const [users, setUsers] = useState([]);

    useEffect( () => {
        load().then(data => setUsers(data));
    })

    return (
        <div className='App'>
            <Profiles users={users}>
        </div>
    )
}
```

TOP

Dependencies

- If you look at the result with the above code, it will run **nonstop**.
- `useEffect` hook accepts dependencies as an array on it's second argument. So if we want the component to render once, we have to give it an empty array.

```
useEffect(() => {
  load().then((data) => {
    setUsers(data);
  });
}, []);
```

TOP

Adding Page Functionality

- Let's make some changes to our app by adding a button and when clicked, it will show different set of data (different page) by emulating turning a page:
- In our `reqres` api, there's a data about pages, we'll use that information to code our app. In order to do that, first we need some modifications to `users.js` file:

```
// users.js file
function load(page = 1) {
    const data = fetch('https://reqres.in/api?page=${page}')
    .then(response => {
        return response.json();
    })
    // .then(results => results.data)
    .catch(err) {
        console.log(err);
    }
    return data;
}

export default load;

//App.js

...
    useEffect( () => {
        load(2).then(data => setUsers(data));
    })

...
```

- Now if we manually change the passing value to load() as 2 or 1, we can see our data is changing on the browser
- Let's actually store this data about the page as a state in the `App` component. currentPage and totalPages store our paging data and has `1` as a default

```
const [currentPage, setCurrentPage] = useState(1);
const [totalPages, setTotalPages] = useState(1);
```

- change our `useEffect`

```
useEffect(() => {
  load(currentPage).then((result) => {
    setCurrentPage(result.page);
    setTotalPages(result.total_pages);
    setUsers(result.data);
  });
});
```

- don't forget to add the missing dependency

```
useEffect(() => {
  load(currentPage).then((result) => {
    setCurrentPage(result.page);
    setTotalPages(result.total_pages);
    setUsers(result.data);
  });
}, [currentPage]);
```

TOP

Paging Component

- Now we have the ability to change the pages, but we have no way of changing it on the app, let's code a button which uses our paging functionality.

```
// App.js

<Paging currentPage={currentPage} totalPage={totalPage} />;

// Paging.js

function Paging({ currentPage, totalPage }) {
  let label = currentPage === totalPage ? "Previous" : "Next";

  return (
    <div>
      <button>{label}</button>
    </div>
  );
}
```

TOP

Add Click Event

- We have a button, but it doesn't really change anything when clicked. Let's add this functionality.

```
// App.js

<Paging currentPage={currentPage} totalPage={totalPage} />;

// Paging.js

function Paging({ currentPage, totalPage }) {
  let label = currentPage === totalPage ? "Previous" : "Next";

  const onClickHandler = () => {
    let newPage;
    if (currentPage === totalPage) {
      newPage = currentPage - 1;
    } else {
      newPage = currentPage + 1;
    }
  };

  return (
    <div>
      <button onClick={onClickHandler}>{label}</button>
    </div>
  );
}
```

- There's a problem with this, because we're actually trying to change props but it doesn't really change the state from our `App` component.
- We have to pass `setCurrentPage` as a prop, so `Paging` component can use it to update the state. Let's fix that.

⚠️ So this is a fundamental idea in React where data and functions that updates this data are passed down from top to bottom.

```
// App.js

<Paging
  currentPage={currentPage}
  totalPage={totalPage}
  setCurrentPage={setCurrentPage}
/>;

// Paging.js

function Paging({ currentPage, totalPage, setCurrentPage }) {
  let label = currentPage === totalPage ? "Previous" : "Next";

  const onClickHandler = () => {
    let newPage;
    if (currentPage === totalPage) {
      newPage = currentPage - 1;
    } else {
      newPage = currentPage + 1;
    }
    setCurrentPage(newPage);
  };

  return (
    <div>
      <button onClick={onClickHandler}>{label}</button>
    </div>
  );
}
```

TOP