- ⏱ 3 hours

Refactor Widgets App

In this project, you will refactor an existing React application that has its components written as Class Components into using Function Components.

Phase 0: Set Up

Start by cloning the solution of the Widgets application starter from this link: https://github.com/appacademy-starters/refactor-widgets.
Run `npm install` to install all the packages.
To start the application, run `npm start` and open http://localhost:3000 to see the widgets displayed. Each section shown on the page is a different component in the `components` folder.

Phase 1: `Folder`

In this phase, you will refactor the `Folder` component from a Class Component into a Function Component. The `Folder` component is rendered inside of the `App` component where it's passed an array of folder objects as props. The `Folder` component renders the tabs of the folder as a sub-component called `Header`. The selected tab is stored as a component state in `Folder` and the content of the selected tab is rendered inside of the `Folder` component.
Create a new `Folder` function component that expects the `folders` prop passed in from its parent component, `App`.
Convert the `currentTab` state in the `Folder` class component into a state variable in the `Folder` function component.
Convert any instances of `this.state` or `this.props` in the `Folder` class component to their respective variables in the `Folder` function component.
Convert the `selectTab` method into a regular function.
Test your conversion out on the browser and be sure to use debugging tools like `debugger` or `console.log` and check the error messages if you run into any issues.

Phase 2: `Auto`

In this phase, you will refactor the `Auto` component from a Class Component into a Function Component. The `Auto` component is rendered inside of the `App` component

where it's passed an array of names as props. The `Auto` component renders a list of names that could match the rendered input's value.
Just like with the previous phase, convert the component props and state in the `Auto` class component to props and state in a function component. Convert the instance methods on the class into regular functions in the function component.

Test your conversion out in the browser.

Phase 3: `Weather`

In this phase, you will refactor the `Weather` component from a Class Component into a Function Component. The `Weather` component is rendered inside of the `App` component but is not passed any props. The `Weather` component fetches from a weather data API endpoint after the component is first rendered. Then, it renders the data it gets from the weather API endpoint.
In order to get the API to accept your HTTP requests, you'll need an API key. API keys should be stored in a `.env` file so create a `.env` file in your root directory and paste the following into that file:

```
REACT_APP_WEATHER_API=b65b43cc09af164f099fe5a807d56972
# REACT_APP_WEATHER_API=8119be3ea48a73ad298d0b280a0d98ad
# REACT_APP_WEATHER_API=e14bad32abd13d701515672995a36e6a
# REACT_APP_WEATHER_API=2a7d6ce7cdd33961673705d6f754d472
# REACT_APP_WEATHER_API=0009c9f9b5283b47fe0b716582e300e0
```

If you encounter a fetch limit it go ahead and comment out the first api key in the `.env` file and comment in the second api key. Repeat the process if you encounter an error again by commenting out the second key and commenting in the third key.
We will be using the api key in the `Weather` component by changing the line in that component from

```
const apiKey = `???`
```

to

```
const apiKey = process.env.REACT_APP_WEATHER_API
```

Just like with the previous phases, convert the component props and state in the `Weather` class component to props and state in a function component.
Convert the instance methods besides `componentDidMount` on the class into regular functions in the function component.
The `componentDidMount` will run after the first render of the component. Convert this into a `useEffect` with an empty dependency array in the function component. `navigator.geolocation.getCurrentPosition` is a method that will invoke the callback function passed in as the first argument with the coordinates of the user on the browser. In this case, `pollWeather` is the function that gets passed the coordinates. You can define the `pollWeather` function in the function component inside the `useEffect` since it will only be used in the `useEffect`.

Test your conversion out in the browser.

Phase 4: `Clock`

In this phase, you will refactor the `Clock` component from a Class Component into a Function Component. The `Clock` component is rendered inside of the `App` component but is not passed any props. The `Clock` component displays the current date time information and will be updated every second.
Just like with the previous phases, convert the component props and state in the `Clock` class component to props and state in a function component.
Convert the instance methods besides the lifecycle methods (`componentDidMount` and `componentWillUnmount`) on the class into regular functions in the function component.
The `componentDidMount` will run after the first render of the component. Convert this into a `useEffect` with an empty dependency array in the function component.
The `componentWillUnmount` takes the interval id returned by the `setInterval` and clears the interval. `componentWillUnmount` will be called right before the component is removed from the component tree. Inside of the `useEffect` function that replaced the `componentDidMount`, create a return function that will clear the interval set by the `useEffect` function. This is a bit tricky, so call for help if you can't solve it within 15 minutes.

Test your conversion out in the browser.

Bonus Phase 1: Convert a previous project into Class components

Now that you've converted Class Components into Function Components, you should test your understanding of Class Components by trying to do the reverse conversion.

Convert a previous project that uses Function Components into Class Components. You can choose any project to convert.

Did you find this lesson helpful?
No
Yes