the shortcut

React Tutorial
Lesson 4

Today's lesson will cover

- Impossible States
- JavaScript Fetch API
- Data Fetching and Re-Fetching
- Memorized Handler
- Explicit Data Fetching
- Third-party libraries
- Async/Await
- Forms
- React Legacy: Class Components and State

Intro to React

Impossible States

- A Impossible states are not easy to spot, which makes them infamous for causing bugs in the UI
- There is nothing wrong with multiple useState hooks in one React component. Be wary once you see multiple state updater functions in a row, however. These conditional states can lead to impossible states, and undesired behavior in the UI
- The impossible state happens when an error occurs for the asynchronous data. The state for the error is set, but the state for the loading indicator isn't revoked

JavaScript Fetch API

- One of the most challenging parts with understanding how JavaScript (JS) works is understanding how to deal with asynchronous requests, which requires and understanding in how promises and callbacks work.
- AJAX stands for Asynchronous JavaScript and XML, and it allows web pages to be updated asynchronously by exchanging data with a web server while the app is running. In short, it essentially means that you can update parts of a web page without reloading the whole page (the URL stays the same)
- **Why Fetch API?** This allows us to perform declarative HTTP requests to a server. The benefit of Fetch API is that it is fully supported by the JS ecosystem, and is also a part of the MDN Mozilla docs.
- The `fetch()` method returns a `Promise` that resolves the `Response` from the `Request` to show the status (successful or not). If you ever get this message `promise {}` in your console log screen, don't panic — it basically means that the `Promise` works, but is waiting to be resolved. So in order to resolve it we need the `.then()` handler (callback) to access the content.
- Clone the template repository

Data Fetching and Re-Fetching with React

- A Impossible states are not easy to spot, which makes them infamous for causing bugs in the UI
- There is nothing wrong with multiple useState hooks in one React component. Be wary once you see multiple state updater functions in a row, however. These conditional states can lead to impossible states, and undesired behavior in the UI
- The impossible state happens when an error occurs for the asynchronous data. The state for the error is set, but the state for the loading indicator isn't revoked
- Re-fetching data each time someone types into the input field isn't optimal because this implementation stresses the API, you might experience errors if you use requests too often.
- For our project, we will use the reliable and informative Hacker News API to request popular tech stories.
    - `API endpoint: http://hn.algolia.com/api/v1/search?query=`

Memoized Handler

- A bit advanced concept based on memorization, Memoized handler can be applied on top of handlers and callback handlers
- React `useCallback` hook allows you to do so

Third party libraries

- Third party libraries usually allow you to handle verbose complicated tasks in much convenient fashion
- For instance alternative to native fetch API is stable library like axios which performs asynchronous requests to remote APIs
- Usually you would have to install the package of another library from npm registry using following command: `npm install *libraryname*` and afterwards you import it in your component

Async/Await

- You'll work with asynchronous data often in React, so it's good to know alternative syntax for handling promises: **async/await**
- To include error handling as before, the try and catch blocks are there to help. If something goes wrong in the try block, the code will jump into the catch block to handle the error. then/catch blocks and async/await with try/catch blocks are both valid for handling asynchronous data in JavaScript and React.

Forms in React

- Forms aren't much different in React than HTML. When we have input fields and a button to submit data from them, we can give our HTML more structure by wrapping it into a form element with a `onSubmit` handler. The button that executes the submission needs only the "submit" type.

React’s Legacy: Class Components and State

- A typical class component is a JavaScript class with a mandatory render method that returns the JSX. The class extends from a React.Component to inherit (class inheritance) all React's component features (e.g. state management for state, lifecycle methods for side-effects). React props are accessed via the class instance (this)
- If no side-effects and no state were used in legacy apps, we'd use a function component instead of a class component. Before 2018--before React Hooks were introduced--React's function components couldn't handle side-effects (useEffecthooks) or state (useState/useReducer hooks). As a result, these components were known as functional stateless components, there only to input props and output JSX
- Before React Hooks, class components were superior to function components because they could be stateful. With a class constructor, we can set an initial state for the component. Also, the component's instance (this) gives access to the current state (this.state) and the component's state updater method (this.setState)

the shortcut

Self Study Assignments

To Dos

- Start working on React or Fullstack JavaScript project, OR complete FCC Frontend Library Certification : Bootstrap, React, Redux, React & Redux, projects; APIs and Microservices Certification : https://www.freecodecamp.org/learn
- Refer to Reactjs official documentation as the best place to learn underlying main concepts of React with examples: https://reactjs.org/docs/hello-world.html