ACADEMIND

User Interaction & State

Making Apps Interactive & Reactive

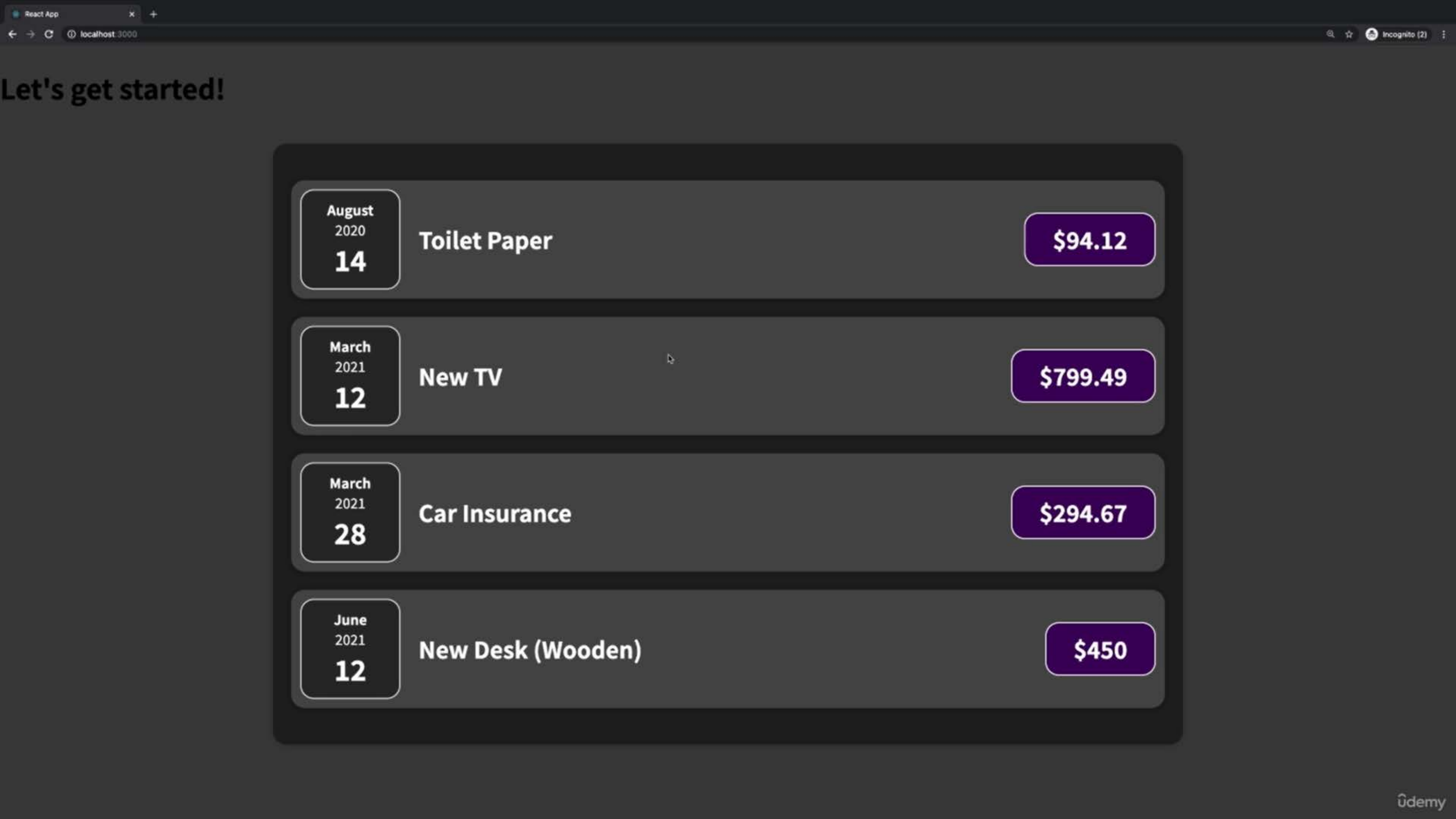
ACADEMIND

React & Components

React allows you to create **re-usable and reactive components** consisting of **HTML and JavaScript** (and CSS)

Declarative Approach

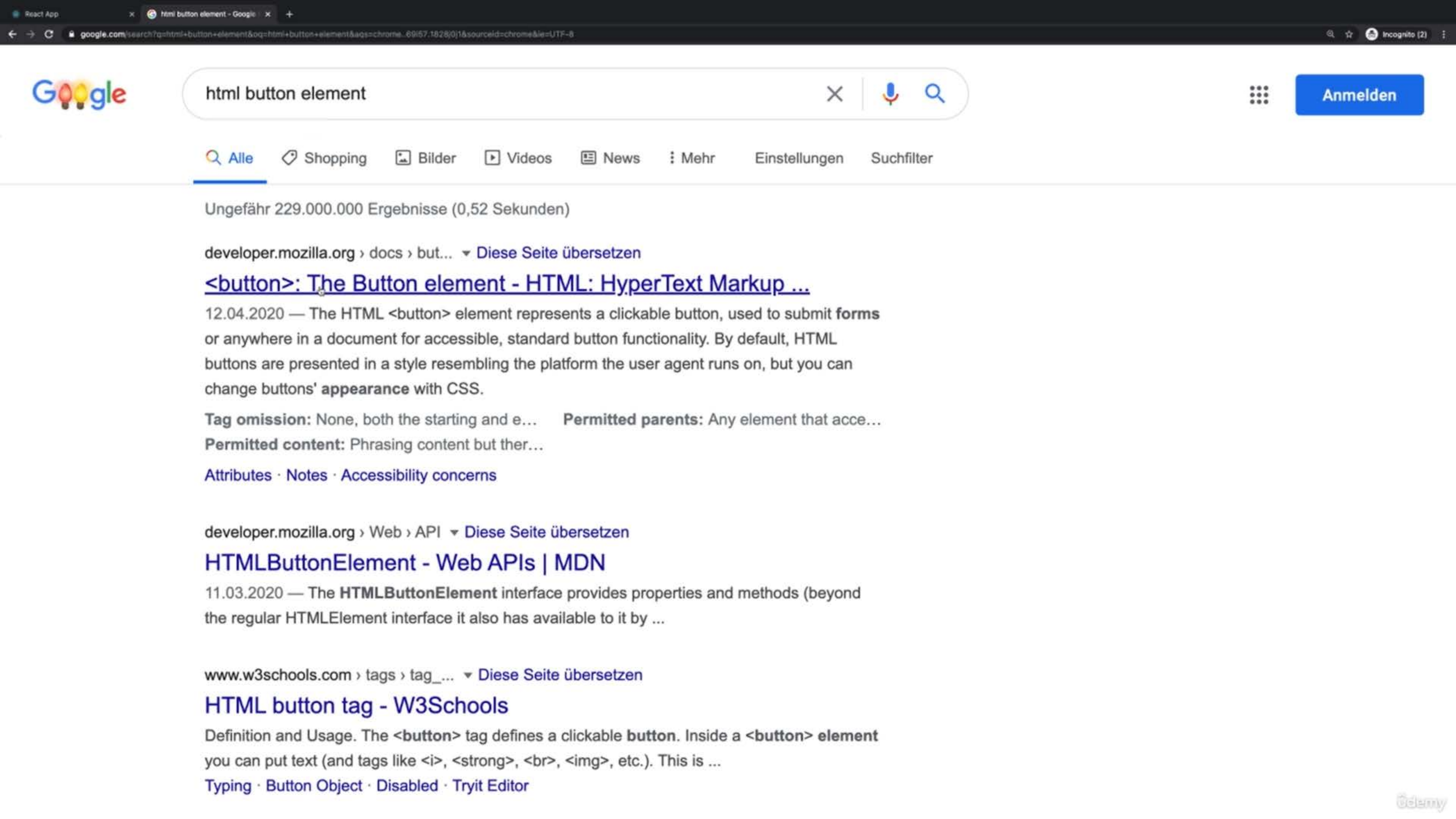
Define the desired target state(s) and let React figure out the actual JavaScript DOM instructions


React App
localhost:3000
Incognito (2)
Let's get started!
August
2020
14
Toilet Paper
$94.12
March
2021
12
New TV
$799.49
March
2021
28
Car Insurance
$294.67
June
2021
12
New Desk (Wooden)
$450
udemy

ACADEMIND

Module Content

- Handling Events
- Updating the UI & Working with “State”
- A Closer Look At Components & State


React App
html button element - Google
google.com/search?q=html+button+element&oq=html+button+element&aqs=chrome..69i57.1828j0j1&sourceid=chrome&ie=UTF-8
Incognito (2)
Google
html button element
Anmelden
Alle
Shopping
Bilder
Videos
News
Mehr
Einstellungen
Suchfilter
Ungefähr 229.000.000 Ergebnisse (0,52 Sekunden)
developer.mozilla.org › docs › but... Diese Seite übersetzen
<button>: The Button element - HTML: HyperText Markup ...
12.04.2020 — The HTML <button> element represents a clickable button, used to submit forms or anywhere in a document for accessible, standard button functionality. By default, HTML buttons are presented in a style resembling the platform the user agent runs on, but you can change buttons' appearance with CSS.
Tag omission: None, both the starting and e… Permitted parents: Any element that acce…
Permitted content: Phrasing content but ther…
Attributes · Notes · Accessibility concerns
developer.mozilla.org › Web › API Diese Seite übersetzen
HTMLButtonElement - Web APIs | MDN
11.03.2020 — The HTMLButtonElement interface provides properties and methods (beyond the regular HTMLElement interface it also has available to it by ...
www.w3schools.com › tags › tag_... Diese Seite übersetzen
HTML button tag - W3Schools
Definition and Usage. The <button> tag defines a clickable button. Inside a <button> element you can put text (and tags like <i>, ,
, <img>, etc.). This is ...
Typing · Button Object · Disabled · Tryit Editor

<button>: The Button element

Related Topics
<button>
<datalist>
<fieldset>
<form>

The **HTML `<button>` element** represents a clickable button, used to submit forms or anywhere in a document for accessible, standard button functionality. By default, HTML buttons are presented in a style resembling the platform the user agent runs on, but you can change buttons' appearance with CSS.

HTML Demo: <button>

Reset

HTML | CSS

```
1 <button class="favorite styled"
2         type="button">
3     Add to favorites
4 </button>
5
```

Output

Add to favorites

```
<button class="favorite styled"
        type="button">
    Add to favorites
</button>
```

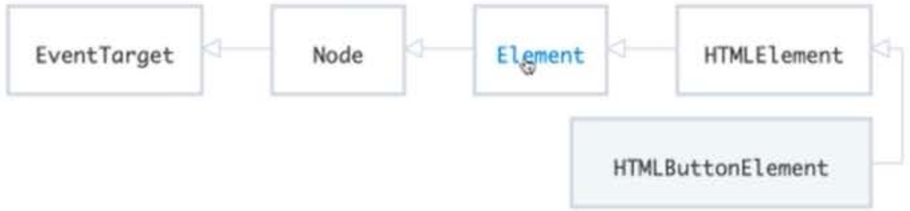
Related Topics

Content categories	Flow content, phrasing content, Interactive content, listed, labelable, and submittable form-associated element, palpable content.
Permitted content	Phrasing content but there must be no Interactive content
Tag omission	None, both the starting and ending tag are mandatory.
Permitted parents	Any element that accepts phrasing content.
Implicit ARIA role	`button`
Permitted ARIA roles	`checkbox`, `link`, `menuitem`, `menuitemcheckbox`, `menuitemradio`, `option`, `radio`, `switch`, `tab`
DOM interface	`HTMLButtonElement`

Attributes

HTMLButtonElement

The **`HTMLButtonElement`** interface provides properties and methods (beyond the regular `HTMLElement` interface it also has available to it by inheritance) for manipulating `<button>` elements.



Properties

Inherits properties from its parent, `HTMLElement`.

`HTMLButtonElement.accessKey`

Is a `DOMString` indicating the single-character keyboard key to give access to the button.

Returns a `DOMString` containing the ARIA role that has been applied to a particular element.

`Element.id`

Is a `DOMString` representing the id of the element.

`Element.innerHTML`

Is a `DOMString` representing the markup of the element's content.

`Element.localName` | Read only

A `DOMString` representing the local part of the qualified name of the element.

`Element.namespaceURI` | Read only

The namespace URI of the element, or `null` if it is no namespace.

> **Note:** In Firefox 3.5 and earlier, HTML elements are in no namespace. In later versions, HTML elements are in the `http://www.w3.org/1999/xhtml` namespace in both HTML and XML trees.

`NonDocumentTypeChildNode.nextElementSibling` | Read only

Is an `Element`, the element immediately following the given one in the tree, or `null` if there's no sibling node.

`Element.outerHTML`

Is a `DOMString` representing the markup of the element including its content. When used as a setter, replaces the element with nodes parsed from the given string.

`Element.part`

Represents the part identifier(s) of the element (i.e. set using the `part` attribute), returned as a `DOMTokenList`.

And if we click on Element, for example,

Element.tabStop

Is a `Boolean` indicating if the element can receive input focus via the tab key.

`Element.tagName` Read only

Returns a `String` with the name of the tag for the given element.

`Element.undoManager` Read only

Returns the `UndoManager` associated with the element.

`Element.undoScope`

Is a `Boolean` indicating if the element is an undo scope host, or not.

> **Note:** DOM Level 3 defined `namespaceURI`, `localName` and `prefix` on the `Node` interface. In DOM4 they were moved to `Element`.
>
> This change is implemented in Chrome since version 46.0 and Firefox since version 48.0.

Properties included from Slotable

The `Element` interface includes the following property, defined on the `Slotable` mixin.

`Slotable.assignedSlot` Read only

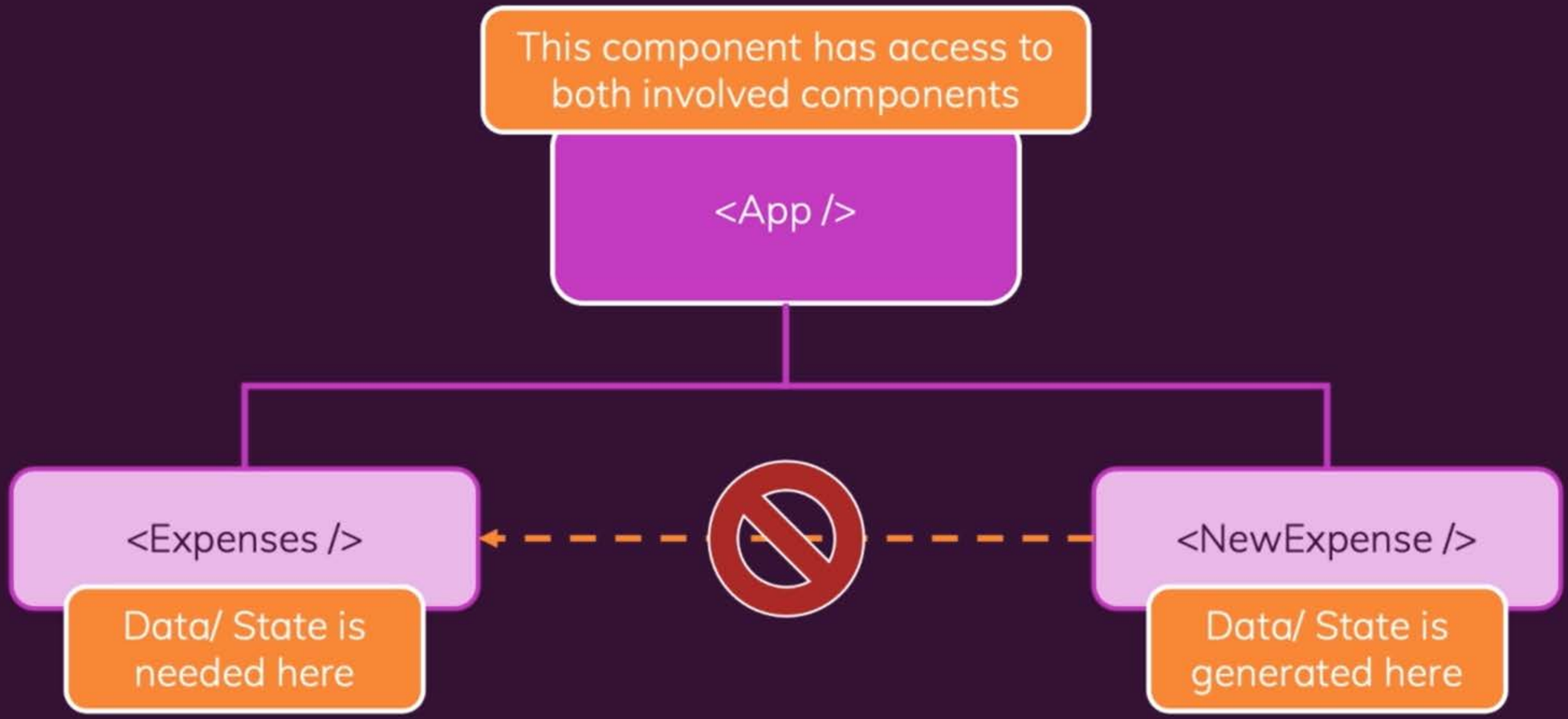
Returns a `HTMLSlotElement` representing the `<slot>` the node is inserted in.

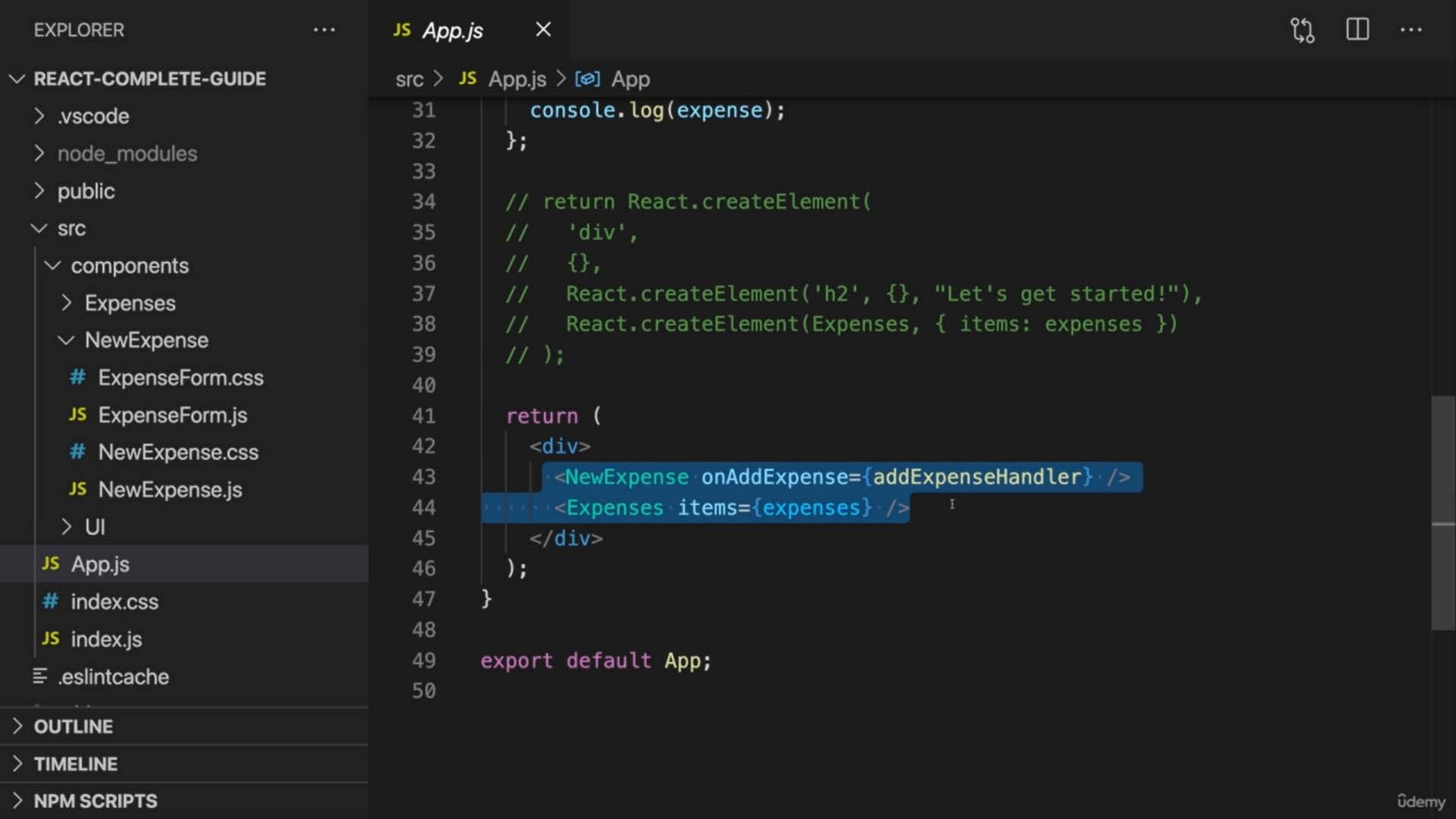
Event handlers

`Element.onfullscreenchange`

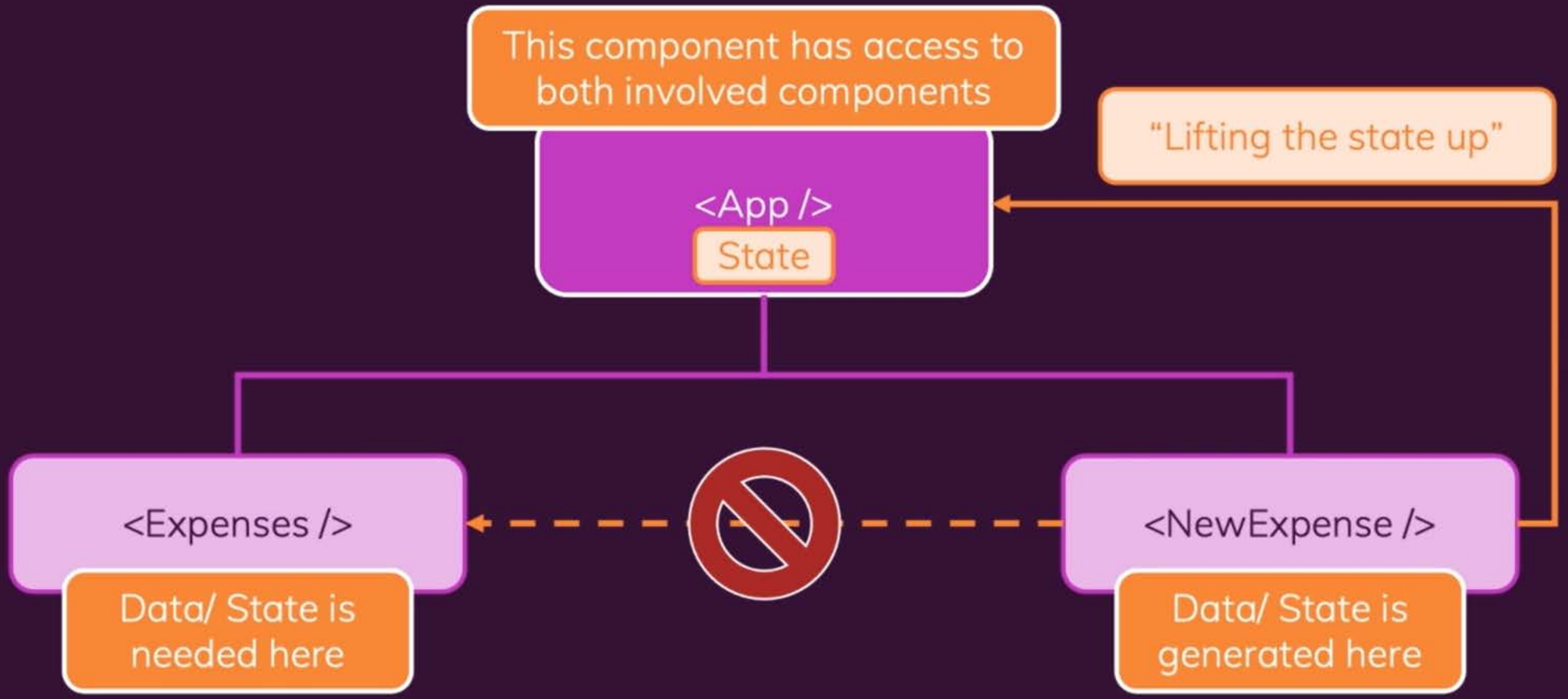
An event handler for the `fullscreenchange` event, which is sent when the element enters or exits full-screen mode. This can be used to watch both for successful expected

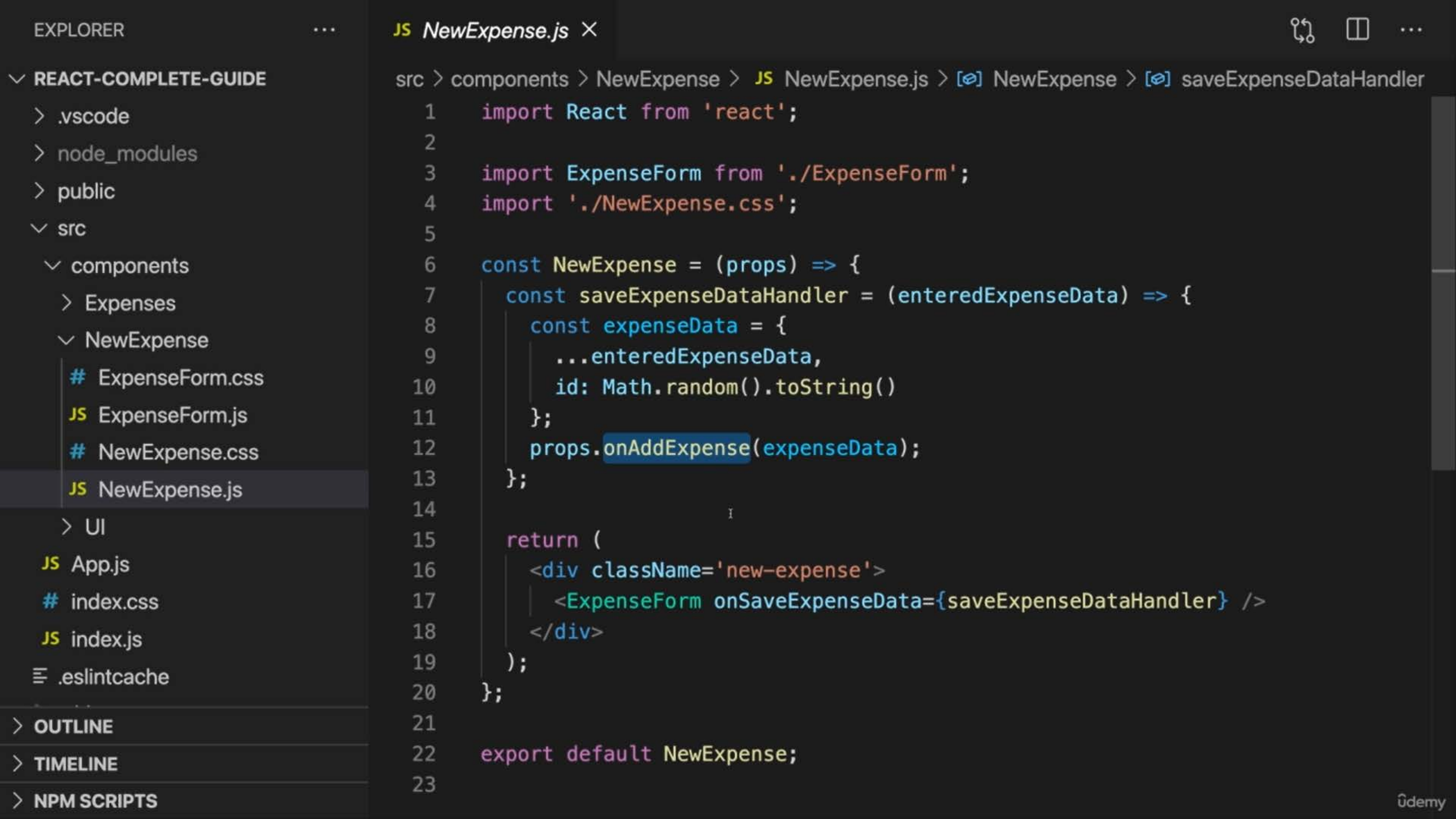
Lifting State Up

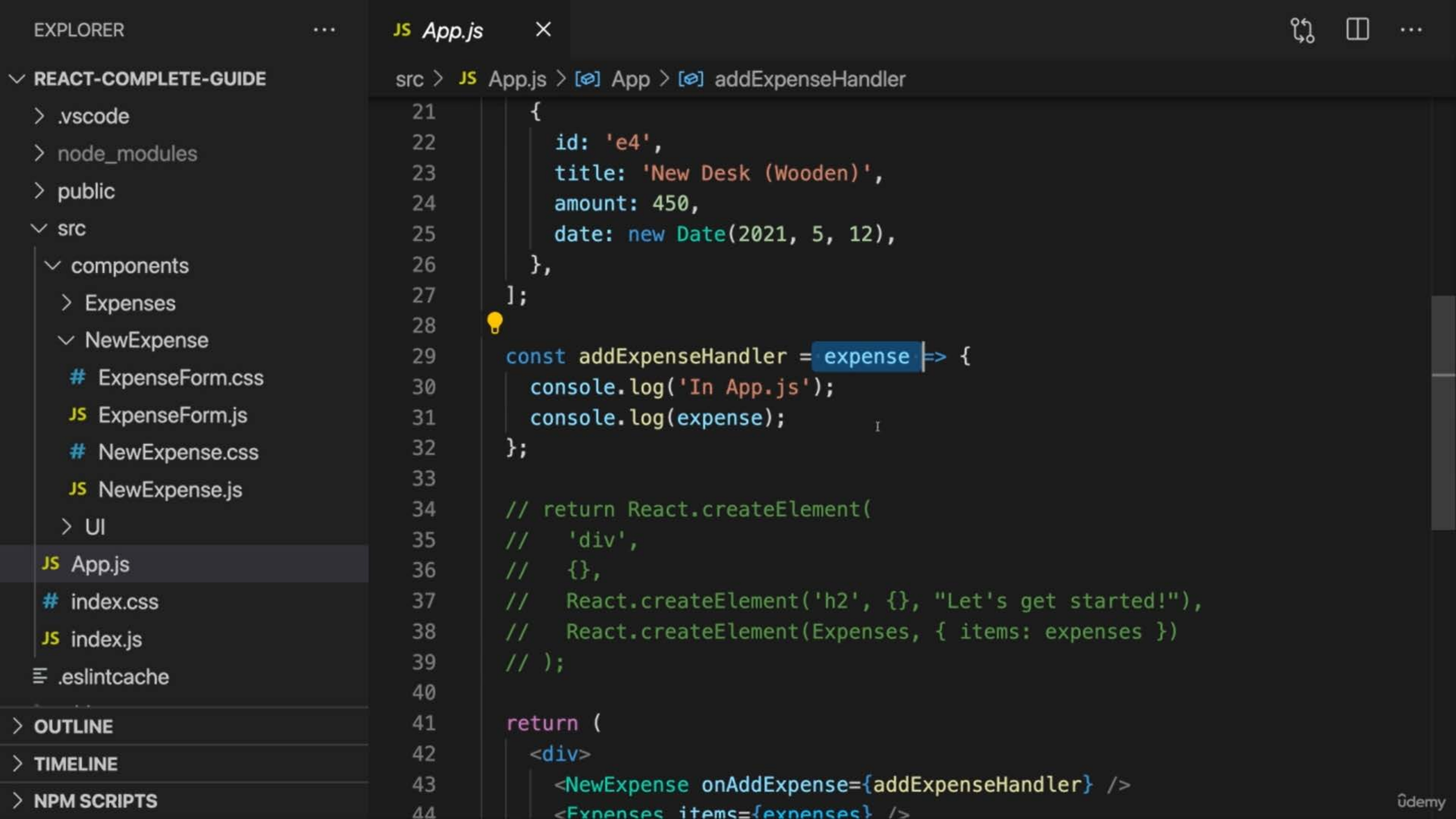



EXPLORER
App.js
REACT-COMPLETE-GUIDE
.vscode
node_modules
public
src
components
Expenses
NewExpense
ExpenseForm.css
ExpenseForm.js
NewExpense.css
NewExpense.js
UI
App.js
index.css
index.js
.eslintcache
OUTLINE
TIMELINE
NPM SCRIPTS
src > App.js > App
31 console.log(expense);
32 };
33
34 // return React.createElement(
35 // 'div',
36 // {},
37 // React.createElement('h2', {}, "Let's get started!"),
38 // React.createElement(Expenses, { items: expenses })
39 //);
40
41 return (
42
43 <NewExpense onAddExpense={addExpenseHandler} />
44 <Expenses items={expenses} />
45
46);
47 }
48
49 export default App;
50
udemy

Lifting State Up




EXPLORER
NewExpense.js
REACT-COMPLETE-GUIDE
.vscode
node_modules
public
src
components
Expenses
NewExpense
ExpenseForm.css
ExpenseForm.js
NewExpense.css
NewExpense.js
UI
App.js
index.css
index.js
.eslintcache
OUTLINE
TIMELINE
NPM SCRIPTS
src > components > NewExpense > NewExpense.js > NewExpense > saveExpenseDataHandler
1 import React from 'react';
2
3 import ExpenseForm from './ExpenseForm';
4 import './NewExpense.css';
5
6 const NewExpense = (props) => {
7 const saveExpenseDataHandler = (enteredExpenseData) => {
8 const expenseData = {
9 ...enteredExpenseData,
10 id: Math.random().toString()
11 };
12 props.onAddExpense(expenseData);
13 };
14
15 return (
16
17 <ExpenseForm onSaveExpenseData={saveExpenseDataHandler} />
18
19);
20 };
21
22 export default NewExpense;
23
udemy


EXPLORER
App.js
REACT-COMPLETE-GUIDE
.vscode
node_modules
public
src
components
Expenses
NewExpense
ExpenseForm.css
ExpenseForm.js
NewExpense.css
NewExpense.js
UI
App.js
index.css
index.js
.eslintcache
OUTLINE
TIMELINE
NPM SCRIPTS
src > App.js > App > addExpenseHandler
21 {
22 id: 'e4',
23 title: 'New Desk (Wooden)',
24 amount: 450,
25 date: new Date(2021, 5, 12),
26 },
27];
28
29 const addExpenseHandler = expense => {
30 console.log('In App.js');
31 console.log(expense);
32 };
33
34 // return React.createElement(
35 // 'div',
36 // {},
37 // React.createElement('h2', {}, "Let's get started!"),
38 // React.createElement(Expenses, { items: expenses })
39 //);
40
41 return (
42
43 <NewExpense onAddExpense={addExpenseHandler} />
udemy

Lifting State Up