# React.js Notes – Part 3

Contents

[React.js Notes – Part 3 1](#_Toc126838226)

[Section 9: Diving Deeper: Working with Fragments, Portals, & "Refs" 1](#_Toc126838227)

[99. Module Introduction 1](#_Toc126838228)

[100. JSX Limitations & Workarounds 1](#_Toc126838229)

[101. Creating a Wrapper Component 8](#_Toc126838230)

[102. React Fragments 9](#_Toc126838231)

[103. Introducing React Portals 11](#_Toc126838232)

[104. Working with Portals 12](#_Toc126838233)

[105. Working with "ref"s 22](#_Toc126838234)

[106. Controlled vs Uncontrolled Components 67](#_Toc126838235)

[Section 10: Advanced: Handling Side Effects, Using Reducers & Using the Context API 69](#_Toc126838236)

[103. Module Introduction 69](#_Toc126838237)

## Section 9: Diving Deeper: Working with Fragments, Portals, & "Refs"

### 99. Module Introduction

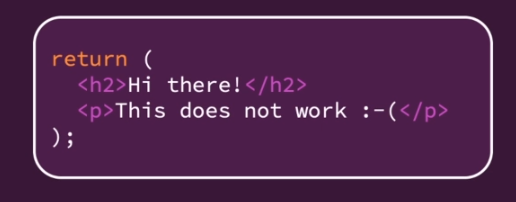
We will look at limitations faced with JSX code and how Fragments can help us overcome those limitations. We'll look at getting a cleaner DOM, better code structure with Portals. We will also look at working with Refs.

### 100. JSX Limitations & Workarounds

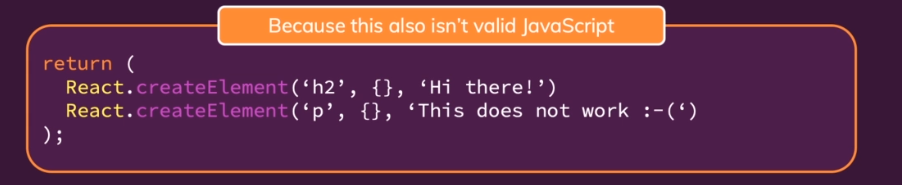
JSX is the code that you return from your components that will in the end be rendered by the real DOM in React.

JSX has limitations.

One limitation is that if we have adjacent root level JSX elements like below, we'll get an error. In React you can't return more than one "root" JSX element (you also can't store more than one "root" JSX element in a variable.

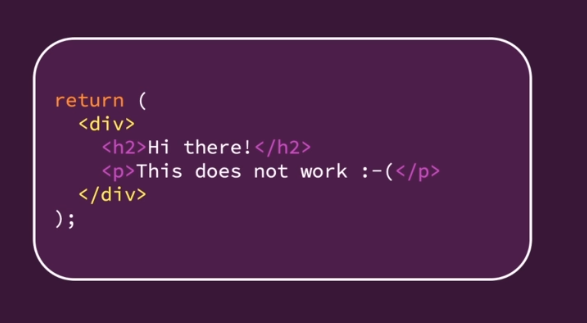


Children elements can be adjacent, but the topmost element can only be one element. This makes sense because in JavaScript we can't return more than one thing.



In the AddUser component we have a wrapping div, and that is a workaround.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <div>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </div>  );  };  export default AddUser; |

You can solve the problem of returning more than one root element by wrapping them in a div.   


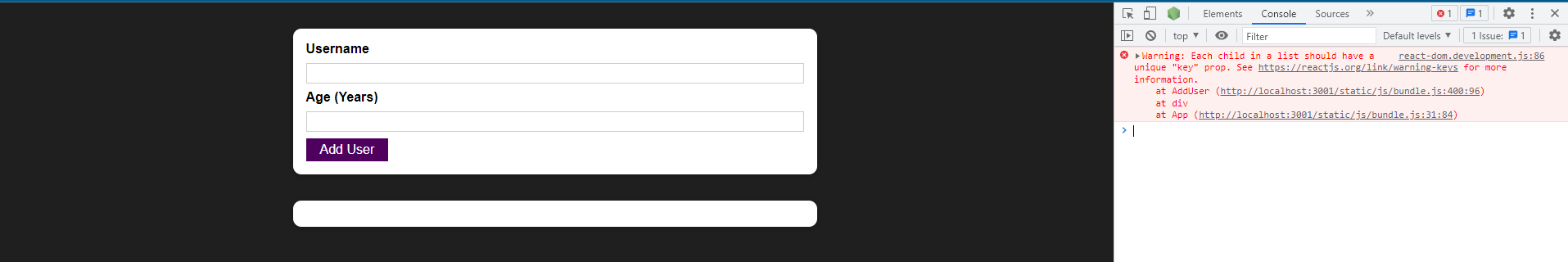
If you wrap the elements in a div, then you are only returning one element. It's similar to returning three numbers by putting them into an array.

**Important:** wrapping element does not have to be a div – ANY element will do the trip.

You could also use a JS array. We could remove our opening and closing div tags and replace with opening and closing square brackets. We would also need to separate the elements with commas and remove the curly braces around the error and ErrorModal because we are no longer inside of JSX here; we are inside of an array.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  **[**  error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )**,**  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  **]**  );  };  export default AddUser; |

We can return an array because React is able to work with arrays of JSX elements. However, we do get a warning in Chrome's developer tools console:



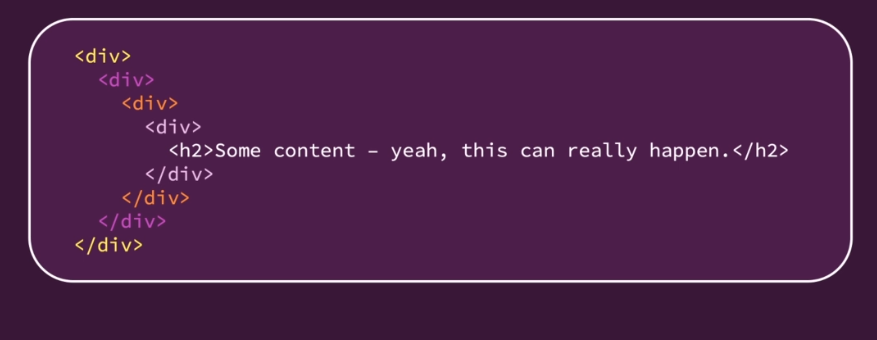
Because we're working with an array of JSX elements, React wants a key on every element. We can simply add a key prop with a custom value to each of the elements.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  [  error && (  <ErrorModal  **key="error-modal"**  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  ),  <Card **key="add-user-card"** className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  ]  );  };  export default AddUser; |

You typically don't use this solution because adding those keys and wrapping it in an array is a little bit cumbersome. It's way easier to simply wrap it in a div like before.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  **<div>**  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  **</div>**  );  };  export default AddUser; |

With the wrapping div or any other wrapping element, a new problem arises. Now we can end up with "<div> Soup" – where you have a real DOM rendered with many nested components and all of those components for various reasons need wrapping divs or other wrapping components, and you have all of these unnecessary divs being rendered in the real DOM even though they're only there because of this limitation in JSX.



In bigger apps, you can easily end up with **tons of unnecessary <div>**s or other elements which add **no semantic meaning or structure** to the page but **are only there because of React's JSX' requirement**.

### 101. Creating a Wrapper Component

We will add a new subfolder in our components folder called "Helpers". We add a Wrapper component. In there we won't import React because we are not going to write any JSX code. Instead we will do this:

|  |
| --- |
| src/components/Helpers/Wrapper.js |
| const Wrapper = props => {  return props.children;  };  export default Wrapper; |

props.children holds all of the content that you are passing between the opening and closing tag of your custom component.

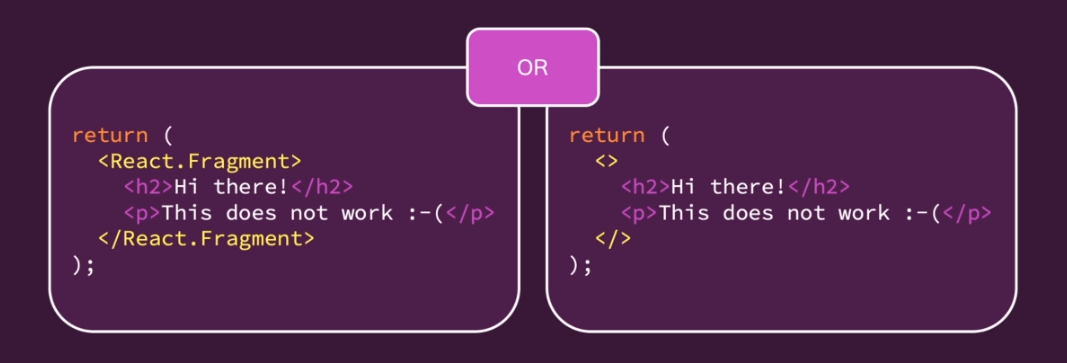
In AddUser we then use our Wrapper by replacing our divs with Wrapper.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  **<Wrapper>**  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  **</Wrapper>**  );  };  export default AddUser; |

The Wrapper is basically an empty component. Everything it does is that it returns props.children.

If we save AddUser, our application works just fine because in AddUser we have a wrapping element. It's a component that won't render anything to the DOM.

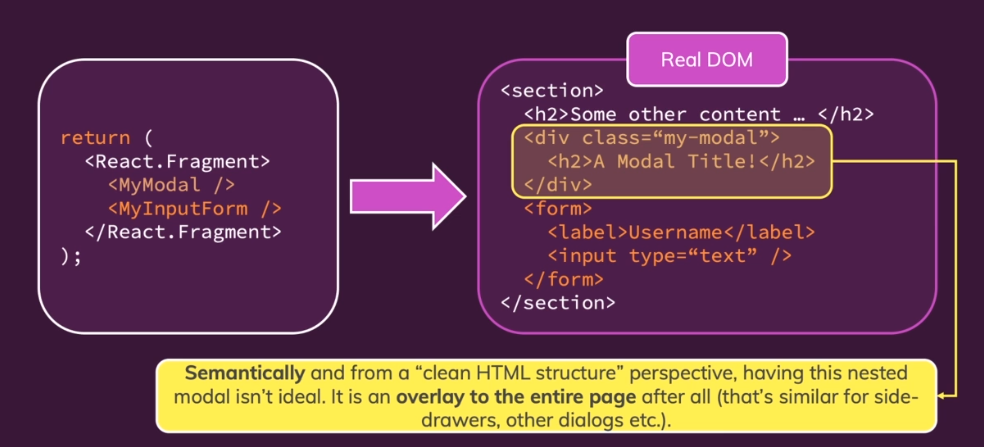
### 102. React Fragments

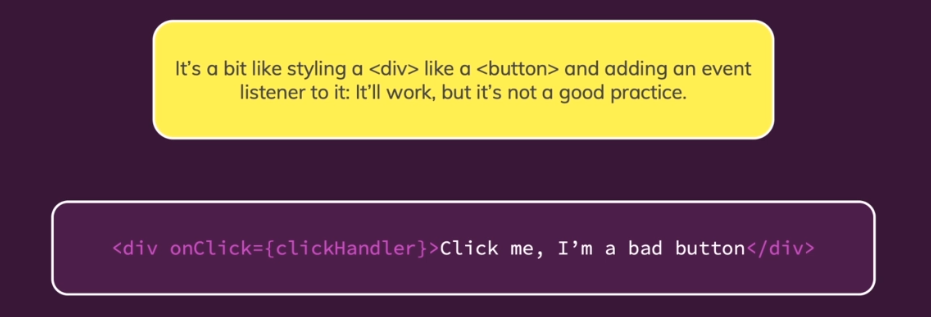
The Wrapper component that we built is not one we need to build because it comes with React. In React it is called the fragment component. You can always use the Fragmenet setup on the left in the screenshot but using the Fragment setup on the right depends on your project setup. 

A fragment is an empty wrapper component that doesn't render any real HTML element to the DOM but fulfills React's JSX requirement. We can use the built in wraper like so in the App component:

|  |
| --- |
| src/App.js |
| import React, { useState } from 'react';  import AddUser from './components/Users/AddUser';  import UsersList from './components/Users/UsersList';  function App() {  const [usersList, setUsersList] = useState([]);  const addUserHandler = (uName, uAge) => {  setUsersList((prevUsersList) => {  return [  ...prevUsersList,  { name: uName, age: uAge, id: Math.random().toString() },  ];  });  };  return (  **<>**  <AddUser onAddUser={addUserHandler} />  <UsersList users={usersList} />  **</>**  );  }  export default App; |

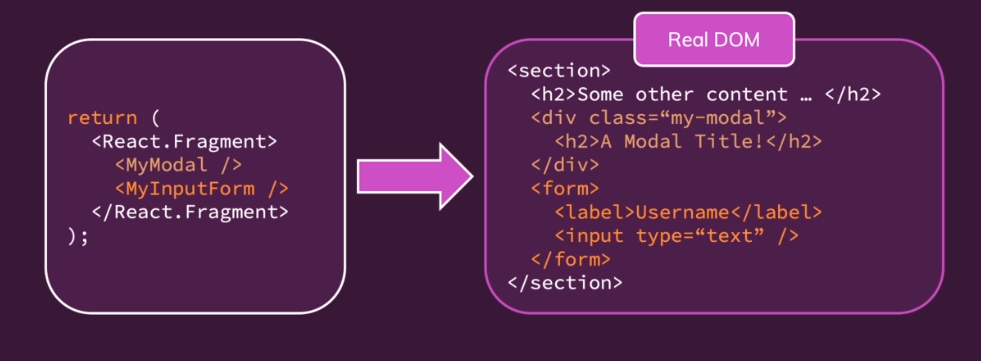
### 103. Introducing React Portals

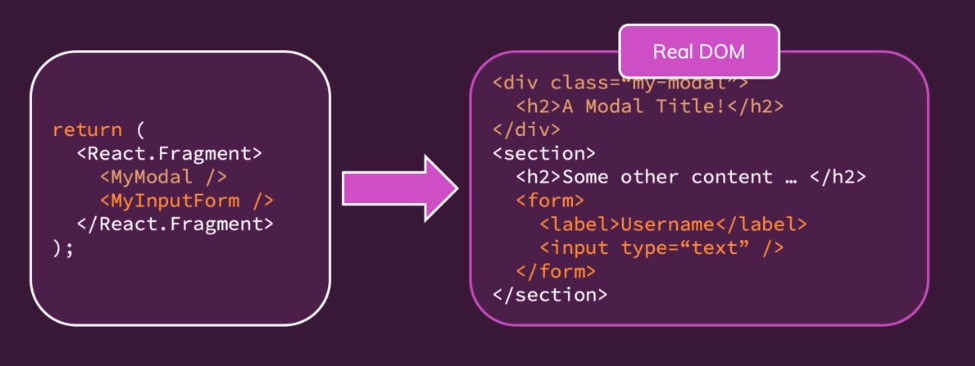




Styling a div like a button is a bad practice for accessibility, bad if a fellow developer has to work on it, and it's just a bad practice in general.

We can use a portal to keep the structure we have on the left in the screenshot below. We want to still be able to write our components the way we want to write them, so that we have no friction when we want to pass data around.



But you still render this differently in the real DOM.

For example, to render the modal html content somewhere else than it would normally go to so that on the left, JSX code re-write hasn't changed, but the rendered HTML code is a bit different from our JSX code. So, the Modal there is not next to the form. You can achieve this with React portals.

### 104. Working with Portals

We of course have an ErrorModal in our project. We can use a fragment in it.

|  |
| --- |
| src/components/UI/ErrorModal.js |
| import React from 'react';  import Card from './Card';  import Button from './Button';  import classes from './ErrorModal.module.css';  const ErrorModal = (props) => {  return (  **<React.Fragment>**  <div className={classes.backdrop} onClick={props.onConfirm} />  <Card className={classes.modal}>  <header className={classes.header}>  <h2>{props.title}</h2>  </header>  <div className={classes.content}>  <p>{props.message}</p>  </div>  <footer className={classes.actions}>  <Button onClick={props.onConfirm}>Okay</Button>  </footer>  </Card>  **</React.Fragment>**  );  };  export default ErrorModal; |

We now should use a portal because the entire ErrorModal component should not be rendered in the place it is being rendered.

Portals need two things

1. You need a place where you want to port the component to.
2. You need to let the component know that it should have a portal to that place.

To mark that place, we go into the "public" folder and then into "index.html". In this file it is common to add a div with an id, which you will then use to identify this place later. You could have ids called "backdrop-root" and "modal-root" and you could create multiple such roots for different kinds of Components that should be portaled there.

|  |
| --- |
| public/index.html |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="utf-8" />  <link rel="icon" href="%PUBLIC\_URL%/favicon.ico" />  <meta name="viewport" content="width=device-width, initial-scale=1" />  <meta name="theme-color" content="#000000" />  <meta  name="description"  content="Web site created using create-react-app"  />  <link rel="apple-touch-icon" href="%PUBLIC\_URL%/logo192.png" />  <!--  manifest.json provides metadata used when your web app is installed on a  user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/  -->  <link rel="manifest" href="%PUBLIC\_URL%/manifest.json" />  <!--  Notice the use of %PUBLIC\_URL% in the tags above.  It will be replaced with the URL of the `public` folder during the build.  Only files inside the `public` folder can be referenced from the HTML.  Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC\_URL%/favicon.ico" will  work correctly both with client-side routing and a non-root public URL.  Learn how to configure a non-root public URL by running `npm run build`.  -->  <title>React App</title>  </head>  <body>  <noscript>You need to enable JavaScript to run this app.</noscript>  **<div id="backdrop-root"></div>**  **<div id="modal-root"></div>**  <div id="root"></div>  <!--  This HTML file is a template.  If you open it directly in the browser, you will see an empty page.  You can add webfonts, meta tags, or analytics to this file.  The build step will place the bundled scripts into the <body> tag.  To begin the development, run `npm start` or `yarn start`.  To create a production bundle, use `npm run build` or `yarn build`.  -->  </body>  </html> |

You could simplify this a bit and have an id of "overlay-root" rather than "modal-root", which will then hold all kinds of overlays – modals, side drawers, and so on. That's what we'll go with.

|  |
| --- |
| public/index.html |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="utf-8" />  <link rel="icon" href="%PUBLIC\_URL%/favicon.ico" />  <meta name="viewport" content="width=device-width, initial-scale=1" />  <meta name="theme-color" content="#000000" />  <meta  name="description"  content="Web site created using create-react-app"  />  <link rel="apple-touch-icon" href="%PUBLIC\_URL%/logo192.png" />  <!--  manifest.json provides metadata used when your web app is installed on a  user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/  -->  <link rel="manifest" href="%PUBLIC\_URL%/manifest.json" />  <!--  Notice the use of %PUBLIC\_URL% in the tags above.  It will be replaced with the URL of the `public` folder during the build.  Only files inside the `public` folder can be referenced from the HTML.  Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC\_URL%/favicon.ico" will  work correctly both with client-side routing and a non-root public URL.  Learn how to configure a non-root public URL by running `npm run build`.  -->  <title>React App</title>  </head>  <body>  <noscript>You need to enable JavaScript to run this app.</noscript>  <div id="backdrop-root"></div>  <div id="**overlay-root**"></div>  <div id="root"></div>  <!--  This HTML file is a template.  If you open it directly in the browser, you will see an empty page.  You can add webfonts, meta tags, or analytics to this file.  The build step will place the bundled scripts into the <body> tag.  To begin the development, run `npm start` or `yarn start`.  To create a production bundle, use `npm run build` or `yarn build`.  -->  </body>  </html> |

We save index.html and then go back to our components.

We now work on the ErrorModal and tell React that it should be portaled somewhere. We can do this by creating a constant, and calling it Backdrop, which will actually be a new Component. We'll add it in this same file because in this app, I only use this Backdrop component in conjunction with my ErrorModal. So we will store both the ErrorModal and Backdrop components in one big file but we could also split it into multiple component files, especially if you would use the backdrop in conjunction with other component files as well. We pass in props and then we return that div that gets my backdrop class: <div className={classes.backdrop} onClick={props.onConfirm} />.

|  |
| --- |
| src/components/UI/ErrorModal.js |
| import React from 'react';  import Card from './Card';  import Button from './Button';  import classes from './ErrorModal.module.css';  **const Backdrop = (props) => {**  **return <div className={classes.backdrop} onClick={props.onConfirm} />**  **};**  const ErrorModal = (props) => {  return (  <React.Fragment>  <div className={classes.backdrop} onClick={props.onConfirm} />  <Card className={classes.modal}>  <header className={classes.header}>  <h2>{props.title}</h2>  </header>  <div className={classes.content}>  <p>{props.message}</p>  </div>  <footer className={classes.actions}>  <Button onClick={props.onConfirm}>Okay</Button>  </footer>  </Card>  </React.Fragment>  );  };  export default ErrorModal; |

We will also add our ModalOverlay component, which gets props, and there I will return the Card and remove Card from where we are currently using it inside of the ErrorModal component.

|  |
| --- |
| src/components/UI/ErrorModal.js |
| import React from 'react';  import Card from './Card';  import Button from './Button';  import classes from './ErrorModal.module.css';  const Backdrop = (props) => {  return <div className={classes.backdrop} onClick={props.onConfirm} />  };  **const ModalOverlay = (props) => {**  **return <Card className={classes.modal}>**  **<header className={classes.header}>**  **<h2>{props.title}</h2>**  **</header>**  **<div className={classes.content}>**  **<p>{props.message}</p>**  **</div>**  **<footer className={classes.actions}>**  **<Button onClick={props.onConfirm}>Okay</Button>**  **</footer>**  **</Card>**  **};**  const ErrorModal = (props) => {  return (  <React.Fragment>  <div className={classes.backdrop} onClick={props.onConfirm} />  </React.Fragment>  );  };  export default ErrorModal; |

I have basically split my modal into two separate Components now – Backdrop and ModalOverlay – because that will make working with portals much easier.

We can also remove the div with the class of backdrop from the ErrorModal component.

|  |
| --- |
| src/components/UI/ErrorModal.js |
| import React from 'react';  import Card from './Card';  import Button from './Button';  import classes from './ErrorModal.module.css';  const Backdrop = (props) => {  return <div className={classes.backdrop} onClick={props.onConfirm} />  };  const ModalOverlay = (props) => {  return <Card className={classes.modal}>  <header className={classes.header}>  <h2>{props.title}</h2>  </header>  <div className={classes.content}>  <p>{props.message}</p>  </div>  <footer className={classes.actions}>  <Button onClick={props.onConfirm}>Okay</Button>  </footer>  </Card>  };  const ErrorModal = (props) => {  return (  <React.Fragment>  </React.Fragment>  );  };  export default ErrorModal; |

What do we do inside of the fragment in the ErrorModal? We can add an expression because we're still inside of JSX code, and we want to call a method that is not actually defined on React but on another library that comes together with React, the react-dom library. You can imagine React as being the library that has all the React features, state management, and so on baked in. And react-dom uses React to bring that logic and these features into the web browser, so making them compatible with working with the DOM.

Put in other words, the react library doesn't care whether you run it in an environment that has a DOM or if you would use it to build a native app. So react-dom is kind of the adapter for React to the browser. And, therefore, since now we're going to portal something into a different place in the real DOM we need to import react-dom.

|  |
| --- |
| src/components/UI/ErrorModal.js |
| import React from 'react';  **import ReactDOM from 'react-dom';**  import Card from './Card';  import Button from './Button';  import classes from './ErrorModal.module.css';  const Backdrop = (props) => {  return <div className={classes.backdrop} onClick={props.onConfirm} />  };  const ModalOverlay = (props) => {  return <Card className={classes.modal}>  <header className={classes.header}>  <h2>{props.title}</h2>  </header>  <div className={classes.content}>  <p>{props.message}</p>  </div>  <footer className={classes.actions}>  <Button onClick={props.onConfirm}>Okay</Button>  </footer>  </Card>  };  const ErrorModal = (props) => {  return (  <React.Fragment>  </React.Fragment>  );  };  export default ErrorModal; |

Inside of the React.Fragment and on ReactDOM you can now call a createPortal method.

|  |
| --- |
| src/components/UI/ErrorModal.js |
| import React from 'react';  import ReactDOM from 'react-dom';  import Card from './Card';  import Button from './Button';  import classes from './ErrorModal.module.css';  const Backdrop = (props) => {  return <div className={classes.backdrop} onClick={props.onConfirm} />  };  const ModalOverlay = (props) => {  return <Card className={classes.modal}>  <header className={classes.header}>  <h2>{props.title}</h2>  </header>  <div className={classes.content}>  <p>{props.message}</p>  </div>  <footer className={classes.actions}>  <Button onClick={props.onConfirm}>Okay</Button>  </footer>  </Card>  };  const ErrorModal = (props) => {  return (  <React.Fragment>  **{ReactDOM.createPortal()}**  </React.Fragment>  );  };  export default ErrorModal; |

The createPortal method takes two arguments. The first one is your React node that should be rendered and here we can render our Backdrop lets say. It's important that we pass JSX, so pass <Backdrop /> rather than Backdrop because your really render it like this: <Backdrop />. We can then forward our onConfirm prop and get access to props.onConfirm, which I need to pass here to make sure that everything still works. I can now set onConfirm in my ErrorModal and forward the function I get on the onConfirm prop to the onClick prop inside of the Backdrop component.

|  |
| --- |
| src/components/UI/ErrorModal.js |
| import React from 'react';  import ReactDOM from 'react-dom';  import Card from './Card';  import Button from './Button';  import classes from './ErrorModal.module.css';  const Backdrop = (props) => {  return <div className={classes.backdrop} onClick={props.onConfirm} />  };  const ModalOverlay = (props) => {  return <Card className={classes.modal}>  <header className={classes.header}>  <h2>{props.title}</h2>  </header>  <div className={classes.content}>  <p>{props.message}</p>  </div>  <footer className={classes.actions}>  <Button onClick={props.onConfirm}>Okay</Button>  </footer>  </Card>  };  const ErrorModal = (props) => {  return (  <React.Fragment>  {ReactDOM.createPortal(**<Backdrop onConfirm={props.onConfirm} />**, )}  </React.Fragment>  );  };  export default ErrorModal; |

The second argument of createPortal is a pointer to that container in the real DOM where this element should be rendered in. And in my case, we of course want to render our backdrop here in that "backdrop-root" component (the div inside of "public/index.html" that has an id of "backdrop-root"). We select the element where it should be rendered to, and for this, we use a DOM API. We use document.getElementById('backdrop-root') for example and get access to the "backdrop-root".

|  |
| --- |
| src/components/UI/ErrorModal.js |
| import React from 'react';  import ReactDOM from 'react-dom';  import Card from './Card';  import Button from './Button';  import classes from './ErrorModal.module.css';  const Backdrop = (props) => {  return <div className={classes.backdrop} onClick={props.onConfirm} />  };  const ModalOverlay = (props) => {  return <Card className={classes.modal}>  <header className={classes.header}>  <h2>{props.title}</h2>  </header>  <div className={classes.content}>  <p>{props.message}</p>  </div>  <footer className={classes.actions}>  <Button onClick={props.onConfirm}>Okay</Button>  </footer>  </Card>  };  const ErrorModal = (props) => {  return (  <React.Fragment>  {ReactDOM.createPortal(<Backdrop onConfirm={props.onConfirm} />, **document.getElementById('backdrop-root')** )}  </React.Fragment>  );  };  export default ErrorModal; |

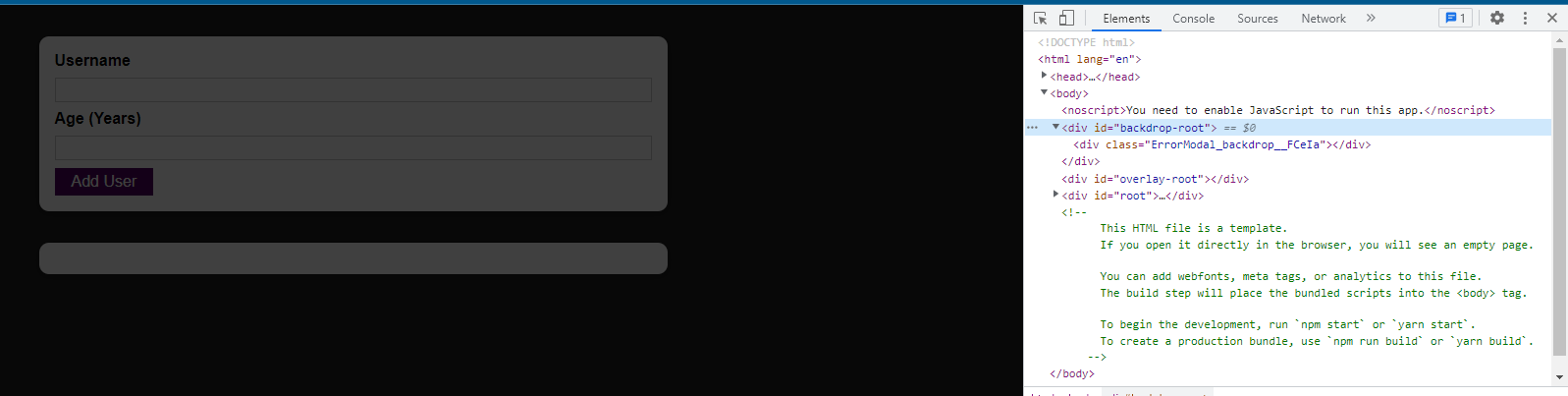
We really get access to a real HTML DOM element, a DOM node here. And we do this with the API that is provided by the browser. document.getElementById has nothing to do with React. WE really get access to a real DOM element with this API.

This is similar how we selected an element in "src/index.js". In "src/index.js" we also rendered the root component with the render method into a place selected with getElementById.

|  |
| --- |
| src/index.js |
| import React from 'react';  import ReactDOM from 'react-dom/client';  import './index.css';  import App from './App';  const root = ReactDOM.createRoot(**document.getElementById('root')**);  root.render(<App />); |

In the ErrorModal we're not rendering an element but inside of an existing application which is already being rendered by React, we portal. We move the HTML content that is about to be rendered into a different place.

If we reload the browser at this point, if we click the backdrop appears, but the modal does not since we haven't added the logic for this yet.



In Chrome's Dev Tools in the Elements tab, we see that in the "backdrop-root", we now have the modal backdrop. And now it will always be there no matter where you would use your error modal in your JSX code. No matter how deeply nested it is in other elements. It will always be here which is of course very close to the body.

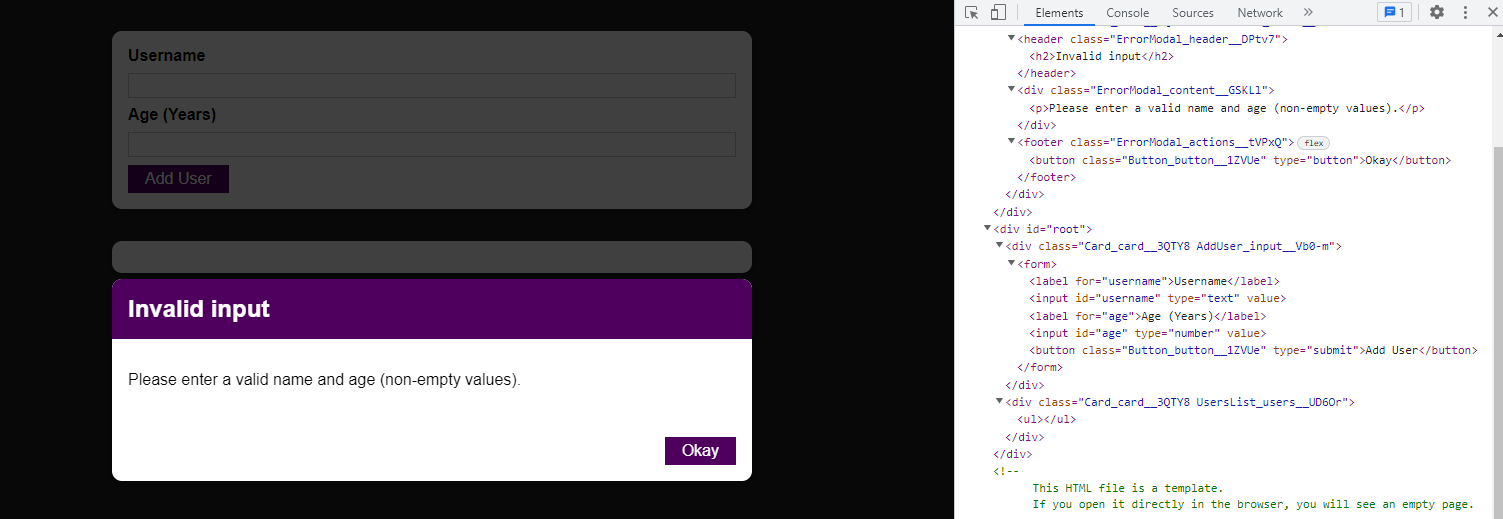
Now we will repeat this for the actual overlay. For that we will add a new expression inside of the ErrorModal next to the expression we just added. Here we use createPortal, but this time we want to render our ModalOverlay component, and my ModalOverlay component needs a bunch of props, which I now of course want to forward. It needs a title, message, and onConfirm.

|  |
| --- |
| src/components/UI/ErrorModal.js |
| import React from 'react';  import ReactDOM from 'react-dom';  import Card from './Card';  import Button from './Button';  import classes from './ErrorModal.module.css';  const Backdrop = (props) => {  return <div className={classes.backdrop} onClick={props.onConfirm} />  };  const ModalOverlay = (props) => {  return <Card className={classes.modal}>  <header className={classes.header}>  <h2>{props.title}</h2>  </header>  <div className={classes.content}>  <p>{props.message}</p>  </div>  <footer className={classes.actions}>  <Button onClick={props.onConfirm}>Okay</Button>  </footer>  </Card>  };  const ErrorModal = (props) => {  return (  <React.Fragment>  {ReactDOM.createPortal(  <Backdrop onConfirm={props.onConfirm} />,  document.getElementById('backdrop-root')  )}  {ReactDOM.createPortal(  **<ModalOverlay**  **title={props.title}**  **message={props.message}**  **onConfirm={props.onConfirm}**  **/>,**  )}  </React.Fragment>  );  };  export default ErrorModal; |

Now just as with the backdrop, we need to let React DOM know where it should render this HTML content where it needs to be rendered. For that, I'll use document.getElementById and select my overlay-root, which was the other div that I created in "public/index.html."

|  |
| --- |
| src/components/UI/ErrorModal.js |
| import React from 'react';  import ReactDOM from 'react-dom';  import Card from './Card';  import Button from './Button';  import classes from './ErrorModal.module.css';  const Backdrop = (props) => {  return <div className={classes.backdrop} onClick={props.onConfirm} />  };  const ModalOverlay = (props) => {  return <Card className={classes.modal}>  <header className={classes.header}>  <h2>{props.title}</h2>  </header>  <div className={classes.content}>  <p>{props.message}</p>  </div>  <footer className={classes.actions}>  <Button onClick={props.onConfirm}>Okay</Button>  </footer>  </Card>  };  const ErrorModal = (props) => {  return (  <React.Fragment>  {ReactDOM.createPortal(  <Backdrop onConfirm={props.onConfirm} />,  document.getElementById('backdrop-root')  )}  {ReactDOM.createPortal(  <ModalOverlay  title={props.title}  message={props.message}  onConfirm={props.onConfirm}  />,  **document.getElementById('overlay-root')**  )}  </React.Fragment>  );  };  export default ErrorModal; |

And with that saved, if we reload and click "Add User," we get the modal popup. With the modal it works and everything is now rendered in the divs where it should be rendered and no longer nested in our other HTML code.



The idea behind a portal really just is that the rendered HTML content is moved somewhere else. ReactDOM.createPortal can be used anywhere where you would otherwise use JSX code. Of course wrap in curly braces because we're using JavaScript code. So wherever you would normally just use a component, you can use createPortal to move that Component's HTML content somewhere else, only in the actual DOM that is being rendered.

### 105. Working with "ref"s

With fragments and portals we end up with semantically more correct HTML code, which makes your app more accessible and makes sure you don't render too many divs unnecessarily, and in general shows that you're a developer who knows what they are doing.

"ref"s are references but the name in React is just ref, so the short form of reference. In their most basic form, refs allow us to get access to other DOM elements and work with them.

In AddUser we have our inputs.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  **<input**  **id="username"**  **type="text"**  **value={enteredUsername}**  **onChange={usernameChangeHandler}**  **/>**  <label htmlFor="age">Age (Years)</label>  **<input**  **id="age"**  **type="number"**  **value={enteredAge}**  **onChange={ageChangeHandler}**  **/>**  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

And we manage what the user enters by simply keeping track of it.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={**usernameChangeHandler**}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={**ageChangeHandler**}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

We simply have our state and with every keystroke we update our state. So with every keystroke, we update the value we get by the user and we store it in our state

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const [enteredUsername, **setEnteredUsername**] = useState('');  const [enteredAge, **setEnteredAge**] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

We feed that state back into the input and then we use that state later on to

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  **<input**  **id="username"**  **type="text"**  **value={enteredUsername}**  **onChange={usernameChangeHandler}**  **/>**  <label htmlFor="age">Age (Years)</label>  **<input**  **id="age"**  **type="number"**  **value={enteredAge}**  **onChange={ageChangeHandler}**  **/>**  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

reset the input but also send it

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  **setEnteredUsername('');**  **setEnteredAge('');**  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

to the place where we need the data.

Updating the state with every keystroke when we only need it when we submit the form seems a bit redundant to me. That's a scenario where refs could help us. Though, refs are not limited to that.

How do refs work? With refs we can setup a connection between an HTML element which is being rendered and our other JavaScript code.

We create a ref which we do with the help of another React hook. We import the useRef hook.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, **useRef** } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

We then simply call useRef in our functional component. Like all React hooks, useRef is only useable inside of functional components. useRef is by default undefined. useRef returns a value that allows us to work with that ref later, so which allows us to work with that element to which we are going to connect it. Here, we will name it nameInputRef because I plan on connecting this ref with that first input that allows us to enter a username.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  **const nameInputRef = useRef();**  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

So here we have nameInputRef and we can create another ref by calling useRef again, and that will be my ageInputRef.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  **const ageInputRef = useRef();**  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

So we now have two refs, but they are not doing anything. They're initialized to be undefined because that is the default.

We can let React know that we want to connect a ref to an HTML element by going to that element to which we want to connect the ref and adding a special prop there, the ref prop. Just like the key prop, the ref prop is a built-in prop, which you can add to any HTML element because you can connect any HTML element to one of your references. You will very often do that for inputs because you want to fetch input data, for example, but you can do it with any element. So we have our ref prop and then we set it to {nameInputRef} as a value.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  **ref={nameInputRef}**  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

The nameInputRef is just that constant which stores this first ref.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  **const nameInputRef = useRef();**  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

So with that, I am connecting this ref

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  **const nameInputRef = useRef();**  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

which I created in this component with the JSX code that is being rendered by that same component.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  **ref={nameInputRef}**  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

And now a connection will be established. The first time react reaches this code and renders this code,

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  **<Wrapper>**  **{error && (**  **<ErrorModal**  **title={error.title}**  **message={error.message}**  **onConfirm={errorHandler}**  **/>**  **)}**  **<Card className={classes.input}>**  **<form onSubmit={addUserHandler}>**  **<label htmlFor="username">Username</label>**  **<input**  **id="username"**  **type="text"**  **value={enteredUsername}**  **onChange={usernameChangeHandler}**  **ref={nameInputRef}**  **/>**  **<label htmlFor="age">Age (Years)</label>**  **<input**  **id="age"**  **type="number"**  **value={enteredAge}**  **onChange={ageChangeHandler}**  **/>**  **<Button type="submit">Add User</Button>**  **</form>**  **</Card>**  **</Wrapper>**  );  };  export default AddUser; |

It will actually set the values stored in nameInputRef to the native DOM element that is

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  **const nameInputRef = useRef();**  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

rendered based on this input.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  **<input**  **id="username"**  **type="text"**  **value={enteredUsername}**  **onChange={usernameChangeHandler}**  **ref={nameInputRef}**  **/>**  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

What will end up inside of nameInputRef in the end will really be a real DOM element later.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={**nameInputRef**}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

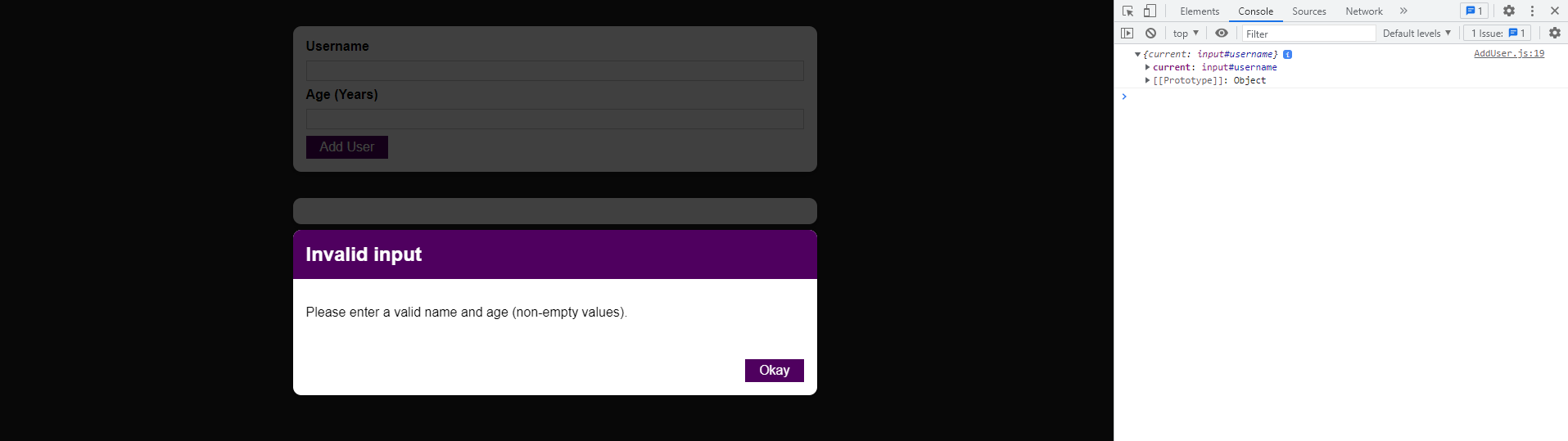
We will do the same for ageInputRef.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  **ref={ageInputRef}**  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

We can try console logging our refs.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  **console.log(nameInputRef);**  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  ref={ageInputRef}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

We see that there is an object being output and that object has a current property.



This nameInputRef being generated here always is an object, which always has a current prop, and the current prop holds the actual value that ref is connected with. By default it is undefined but

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  **const nameInputRef = useRef();**  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  console.log(nameInputRef);  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  ref={ageInputRef}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

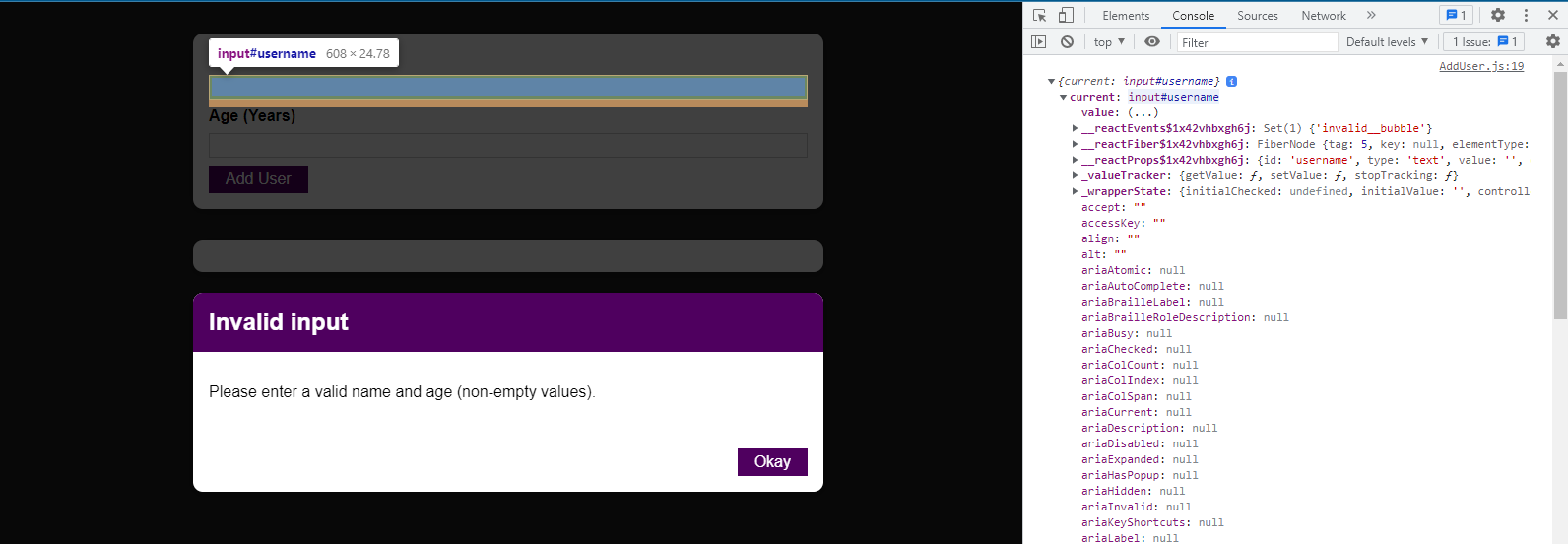
as soon as this code ran

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  console.log(nameInputRef);  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  **<Wrapper>**  **{error && (**  **<ErrorModal**  **title={error.title}**  **message={error.message}**  **onConfirm={errorHandler}**  **/>**  **)}**  **<Card className={classes.input}>**  **<form onSubmit={addUserHandler}>**  **<label htmlFor="username">Username</label>**  **<input**  **id="username"**  **type="text"**  **value={enteredUsername}**  **onChange={usernameChangeHandler}**  **ref={nameInputRef}**  **/>**  **<label htmlFor="age">Age (Years)</label>**  **<input**  **id="age"**  **type="number"**  **value={enteredAge}**  **onChange={ageChangeHandler}**  **ref={ageInputRef}**  **/>**  **<Button type="submit">Add User</Button>**  **</form>**  **</Card>**  **</Wrapper>**  );  };  export default AddUser; |

because of this ref prop,

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  **const nameInputRef = useRef();**  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  console.log(nameInputRef);  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={**nameInputRef**}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  ref={ageInputRef}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

the nameInputRef is connected to that input, and hence it is actually the input, which is being stored as a value in the current prop. What’s being stored here really is the actual DOM node, which you could now manipulate and do all kinds of things with. It is recommended that you don’t manipulate it. Your DOM should only really be manipulated by React. You’re using React to let it do all the heavy lifting. Reading data from the input doesn’t sound too bad because you’re not changing anything with that. You’re just reading data.



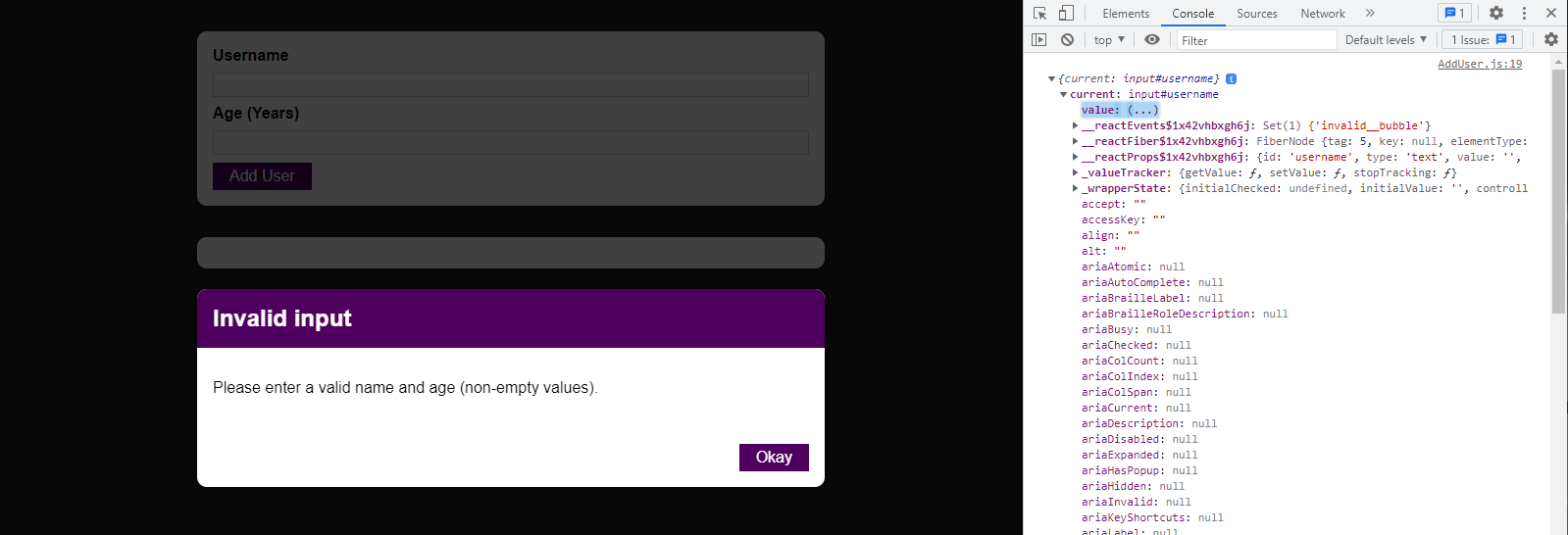
So here, instead of logging this,

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  **console.log(nameInputRef);**  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  ref={ageInputRef}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

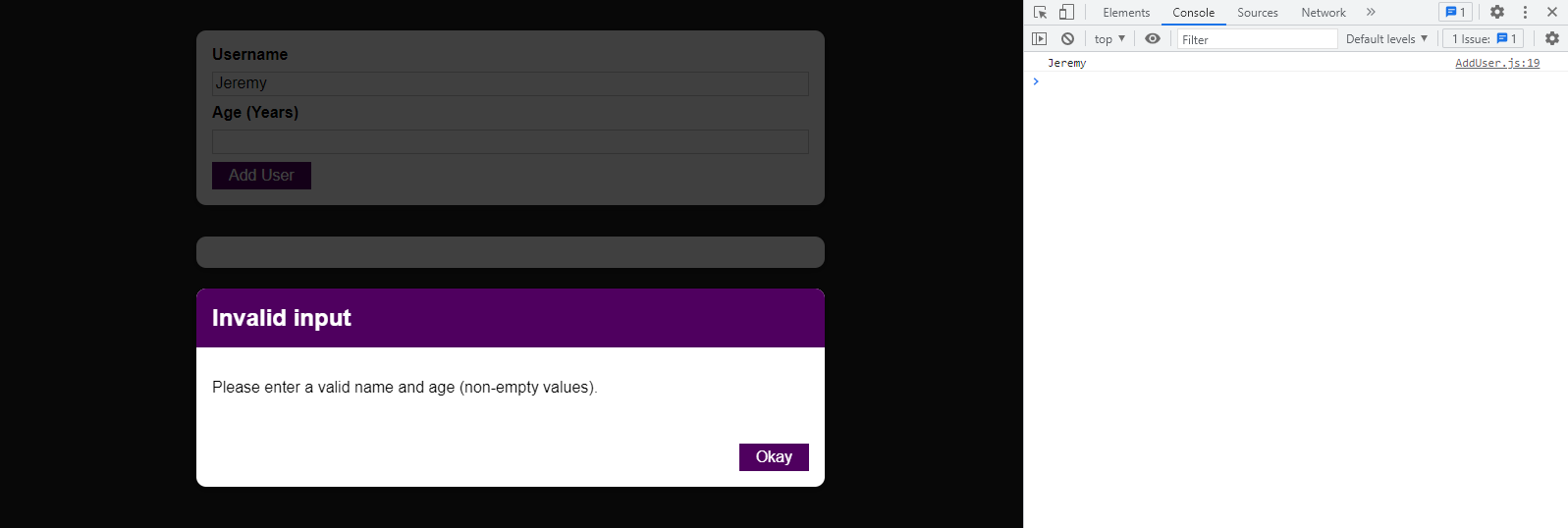
we can, of course, read current.value. Now current refers to the value stored and the value stored is the input element and every input element has a value property in JavaScript.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  console.log(nameInputRef**.current.value**);  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  ref={ageInputRef}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

We can actually see the value property here:



If we now save our code, reload the browser and enter “Jeremy” here, we see “Jeremy” logged to the console:



That means we can get access to the values stored in the element without having to log every keystroke. We don’t need state for this. We can just read it when the submit button is pressed.

That means we can now replace console.log maybe with a helper constant, which we use inside of addUserHandler. We’ll name the constant “enteredName” and simply store my nameInputRef.current.value inside of that.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  **const enteredName =** nameInputRef.current.value;  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  ref={ageInputRef}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

We can do the same for the age. We can store ageInputRef.current.value inside of a constant called enteredUserAge.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  const enteredName = nameInputRef.current.value;  **const entereUserdAge = ageInputRef.current.value;**  if (enteredUsername.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  ref={ageInputRef}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

Now we can check enteredName here:

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  const enteredName = nameInputRef.current.value;  const enteredUserAge = ageInputRef.current.value;  if (**enteredName**.trim().length === 0 || enteredAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  ref={ageInputRef}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

and check enteredUserAge here:

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  const enteredName = nameInputRef.current.value;  const enteredUserAge = ageInputRef.current.value;  if (enteredName.trim().length === 0 || **enteredUserAge**.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  ref={ageInputRef}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

And we also check enteredUserAge here in this condition, so always the values we retrieve from the refs.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  const enteredName = nameInputRef.current.value;  const enteredUserAge = ageInputRef.current.value;  if (enteredName.trim().length === 0 || enteredUserAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+**enteredUserAge** < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredUsername, enteredAge);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  ref={ageInputRef}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

We also use enteredName and enteredUserAge here to submit it to onAddUser.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  const enteredName = nameInputRef.current.value;  const enteredUserAge = ageInputRef.current.value;  if (enteredName.trim().length === 0 || enteredUserAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredUserAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(**enteredName**, **enteredUserAge**);  setEnteredUsername('');  setEnteredAge('');  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  ref={ageInputRef}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

We should no longer reset these inputs by resetting the state here because we’re not using the state to get our values anymore.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  const enteredName = nameInputRef.current.value;  const enteredUserAge = ageInputRef.current.value;  if (enteredName.trim().length === 0 || enteredUserAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredUserAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredName, enteredUserAge);  **setEnteredUsername(''); // should no longer be resetting this input**  **setEnteredAge(''); // should no longer be resetting this input**  };  const usernameChangeHandler = (event) => {  setEnteredUsername(event.target.value);  };  const ageChangeHandler = (event) => {  setEnteredAge(event.target.value);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  ref={ageInputRef}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

We still have those listeners but we actually don’t really need those values anymore because we are using refs to get the values.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [enteredUsername, setEnteredUsername] = useState('');  const [enteredAge, setEnteredAge] = useState('');  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  const enteredName = nameInputRef.current.value;  const enteredUserAge = ageInputRef.current.value;  if (enteredName.trim().length === 0 || enteredUserAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredUserAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredName, enteredUserAge);  setEnteredUsername(''); // should no longer be resetting this input  setEnteredAge(''); // should no longer be resetting this input  };  **const usernameChangeHandler = (event) => {**  **setEnteredUsername(event.target.value);**  **};**  **const ageChangeHandler = (event) => {**  **setEnteredAge(event.target.value);**  **};**  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  value={enteredUsername}  onChange={usernameChangeHandler}  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  value={enteredAge}  onChange={ageChangeHandler}  ref={ageInputRef}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

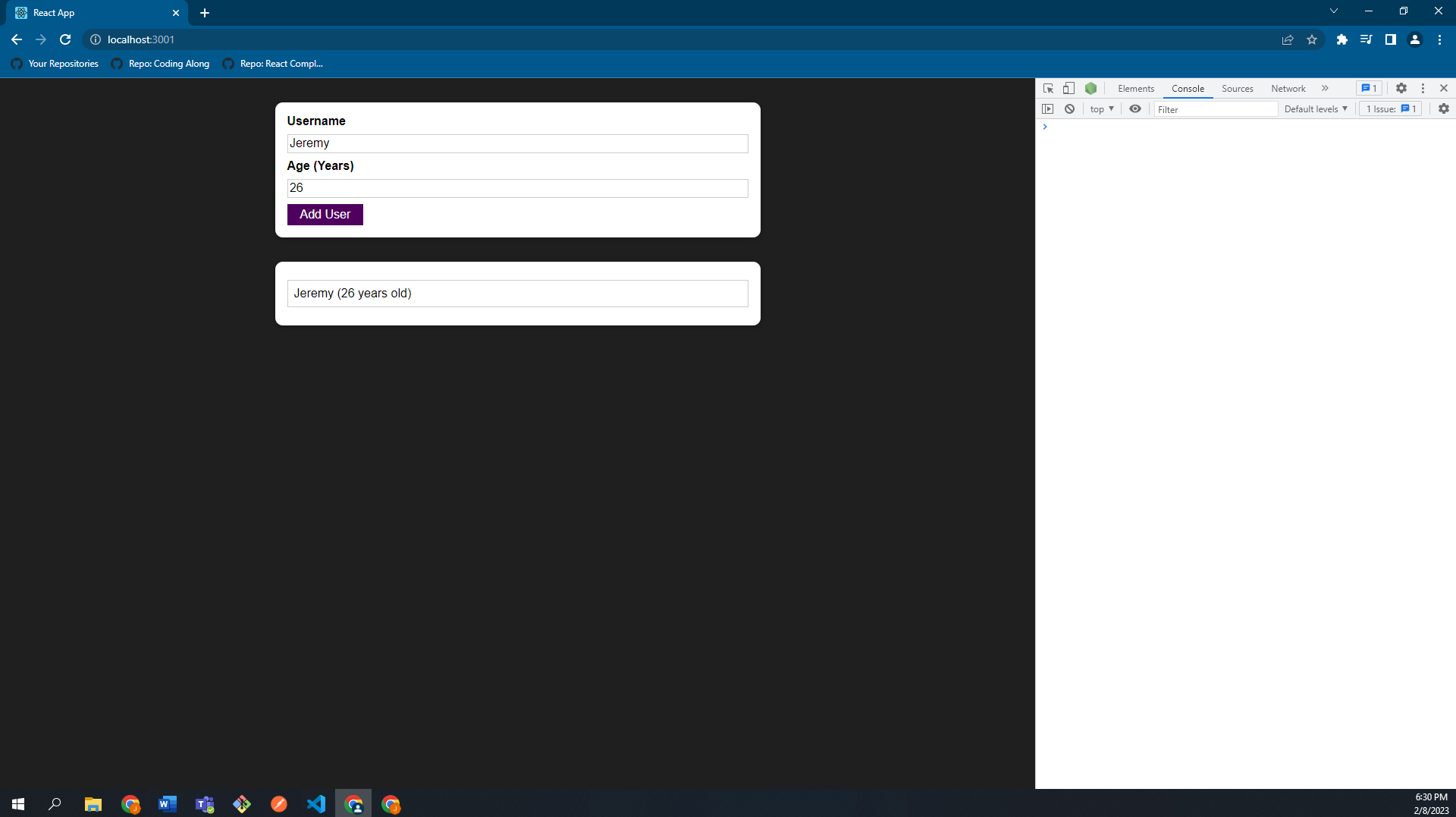
So we can get rid of these states where we listen to every keystroke, get rid of the state updating functions, and get rid of the two handlers. Also, get rid of the value property and the onChange property.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  ~~const [enteredUsername, setEnteredUsername] = useState('');~~  ~~const [enteredAge, setEnteredAge] = useState('');~~  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  const enteredName = nameInputRef.current.value;  const enteredUserAge = ageInputRef.current.value;  if (enteredName.trim().length === 0 || enteredUserAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredUserAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredName, enteredUserAge);  ~~setEnteredUsername(''); // should no longer be resetting this input~~  ~~setEnteredAge(''); // should no longer be resetting this input~~  };  ~~const usernameChangeHandler = (event) => {~~  ~~setEnteredUsername(event.target.value);~~  ~~};~~  ~~const ageChangeHandler = (event) => {~~  ~~setEnteredAge(event.target.value);~~  ~~};~~  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  ~~value={enteredUsername}~~  ~~onChange={usernameChangeHandler}~~  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  ~~value={enteredAge}~~  ~~onChange={ageChangeHandler}~~  ref={ageInputRef}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

After getting rid of state-related items in AddUser:

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  const enteredName = nameInputRef.current.value;  const enteredUserAge = ageInputRef.current.value;  if (enteredName.trim().length === 0 || enteredUserAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredUserAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredName, enteredUserAge);  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  ref={ageInputRef}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

With that, we’ve shortened up the code quite a bit, and we’re relying on refs to read the values. Hence, if I reload the browser and add “Jeremy” and “26”, we can still add that user.



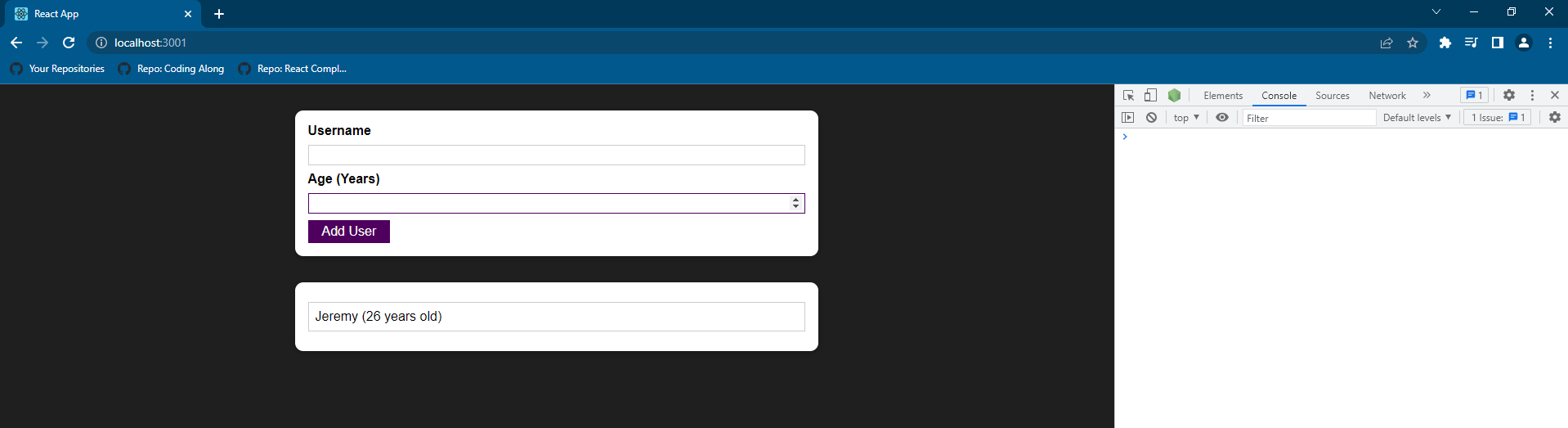
During the process we lost our resetting logic. To bring the resetting logic back, we have two options. We can switch back to the state-based solution, which is not bad but we want to show refs here. Or we can do something which we rarely do but is okay here in the context of an input field value which you want to reset. You can manipulate the DOM without react, and yes, he said that you typically shouldn’t do that, but if it is just resetting the value entered by the user, it is something you can consider doing. We can do the following to reset the values:

nameInputRef.current.value = '';

ageInputRef.current.value = '';

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  const enteredName = nameInputRef.current.value;  const enteredUserAge = ageInputRef.current.value;  if (enteredName.trim().length === 0 || enteredUserAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredUserAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredName, enteredUserAge);  **nameInputRef.current.value = '';**  **ageInputRef.current.value = '';**  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input  id="username"  type="text"  ref={nameInputRef}  />  <label htmlFor="age">Age (Years)</label>  <input  id="age"  type="number"  ref={ageInputRef}  />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

If we now save this and reload our browser, the inputs will be cleared.



Again, we should rarely use refs to manipulate the DOM. Here we’re not really manipulating a DOM. We’re not adding a new element or changing a CSS class. We’re just changing what the user entered. You could always argue to not do that and change back to using the state-based solution.

In general, the question is not whether refs or state is better. You can use either of the two. You will sometimes have use cases where you want to just quickly read a value, for example. And if you only want to read a value and never plan on changing anything, then you don’t really need state because just using state as a keylogger is not that great. It’s a lot of unnecessary code and work. So if you just want to read a value, refs are probably better.

### 106. Controlled vs Uncontrolled Components

This approach of using refs to interact with DOM elements, specifically with input elements, also has a special name. The approach of accessing values with a ref is known as an uncontrolled component. The below inputs (bolded in the below code snippet) would be considered uncontrolled components.

|  |
| --- |
| src/components/Users/AddUser.js |
| import React, { useState, useRef } from 'react';  import Card from '../UI/Card';  import Button from '../UI/Button';  import ErrorModal from '../UI/ErrorModal';  import Wrapper from '../Helpers/Wrapper';  import classes from './AddUser.module.css';  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [error, setError] = useState();  const addUserHandler = (event) => {  event.preventDefault();  const enteredName = nameInputRef.current.value;  const enteredUserAge = ageInputRef.current.value;  if (enteredName.trim().length === 0 || enteredUserAge.trim().length === 0) {  setError({  title: 'Invalid input',  message: 'Please enter a valid name and age (non-empty values).',  });  return;  }  if (+enteredUserAge < 1) {  setError({  title: 'Invalid age',  message: 'Please enter a valid age (> 0).',  });  return;  }  props.onAddUser(enteredName, enteredUserAge);  nameInputRef.current.value = '';  ageInputRef.current.value = '';  };  const errorHandler = () => {  setError(null);  };  return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  **<input**  **id="username"**  **type="text"**  **ref={nameInputRef}**  **/>**  <label htmlFor="age">Age (Years)</label>  **<input**  **id="age"**  **type="number"**  **ref={ageInputRef}**  **/>**  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  );  };  export default AddUser; |

They are considered uncontrolled because they’re internal state, so the value that is reflected in them is not controlled by react. We rely on the default behavior of the input where of course a user is able to enter something, and that entered value is reflected. And then we just fetch it, with a react feature, but we don’t feed data back into the inputs.

When I set a new value to those inputs using the following:

nameInputRef.current.value = '';

ageInputRef.current.value = '';

with this workaround, we’re absolutely not using react here. We are using the ref, but in the end, we get access to the native DOM element with that and then I just use the regular DOM api for setting the value of a dom node of an input dom node. That is why the input is considered uncontrolled because we’re not controlling the state of the input element with react.

You can talk about uncontrolled and controlled components also in the context of other components, but most commonly, we have this scenario when we talk about input components, about form components in general, because those components tend to have some internal state natively by the browser, an input element is configured to take user input and save and reflect it. And when we then work with those components in a react app, we want to connect our react state to that. That’s why we typically have this controlled vs. uncontrolled thing when we work with input components in react.

The approach where we manage our state and update that state on every keystroke and we feed that state back into the input with that state back into the input with the value prop. This is the controlled approach. We would then say that those input components are controlled components because their internal state is controlled by react.

## Section 10: Advanced: Handling Side Effects, Using Reducers & Using the Context API

### 108. Module Introduction

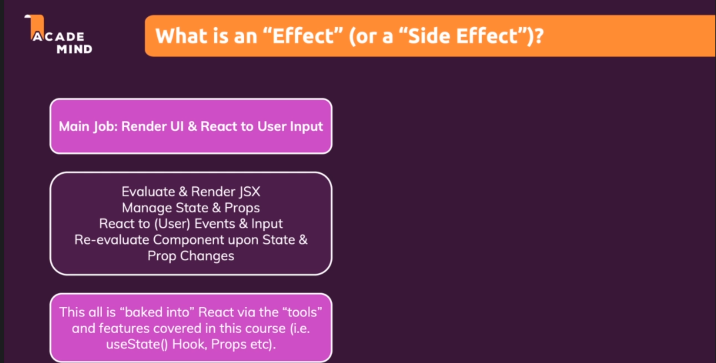
We’re going to take a look at 3 very important concepts that you need to know as a react developer: Effects, Reducers & Context. These features are more advanced (does not mean they are difficult), meaning that you need to know the concepts that came before this.

This module will work with what effects are and how you can work with side effects. We will look at Managing more Complex State with Reducers.

We will also look at Managing App-Wide or Component-Wide State with Context (which is a concept built into react to make sharing state and state updates across components easier).

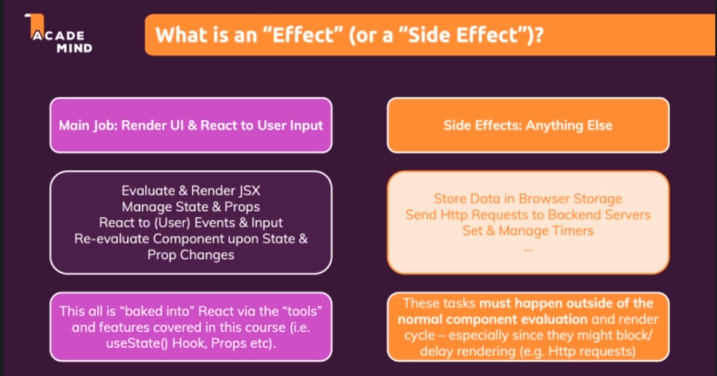
### 109. What are “Side Effects” & Introducing useEffect

Our components in our React app and our React app as a whole and also the react library itself has one main job to Render the UI & React to user input and re-render the UI when it’s needed.

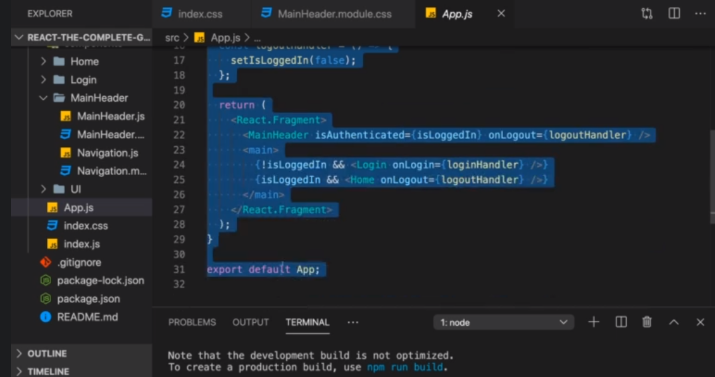


What is an “Effect” (or a “Side Effect”)?

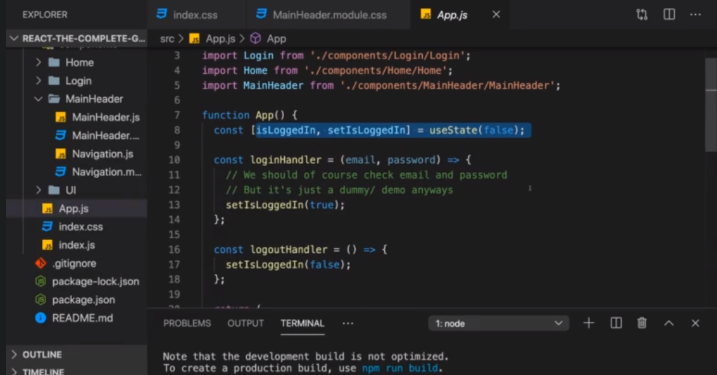
Side Effects are anything else that might be happening in your application. A good example is sending an HTTP request or storing something in browser storage, in local storage, for example. You could also think about timers or intervals which you set in your code. These are all tasks that you often consider in your application. These tasks are all not related to bringing something onto the screen. At least not directly. You may might be sending an http request to then draw something onto the screen once you get the response but sending the request itself and handling potential errors and so on is not something you need React for. It’s not something react cares about. These tasks must happen outside of the normal component evaluation, so outside of your normal component function.



If we were to send in an HTTP request here, then we would send this request whenever



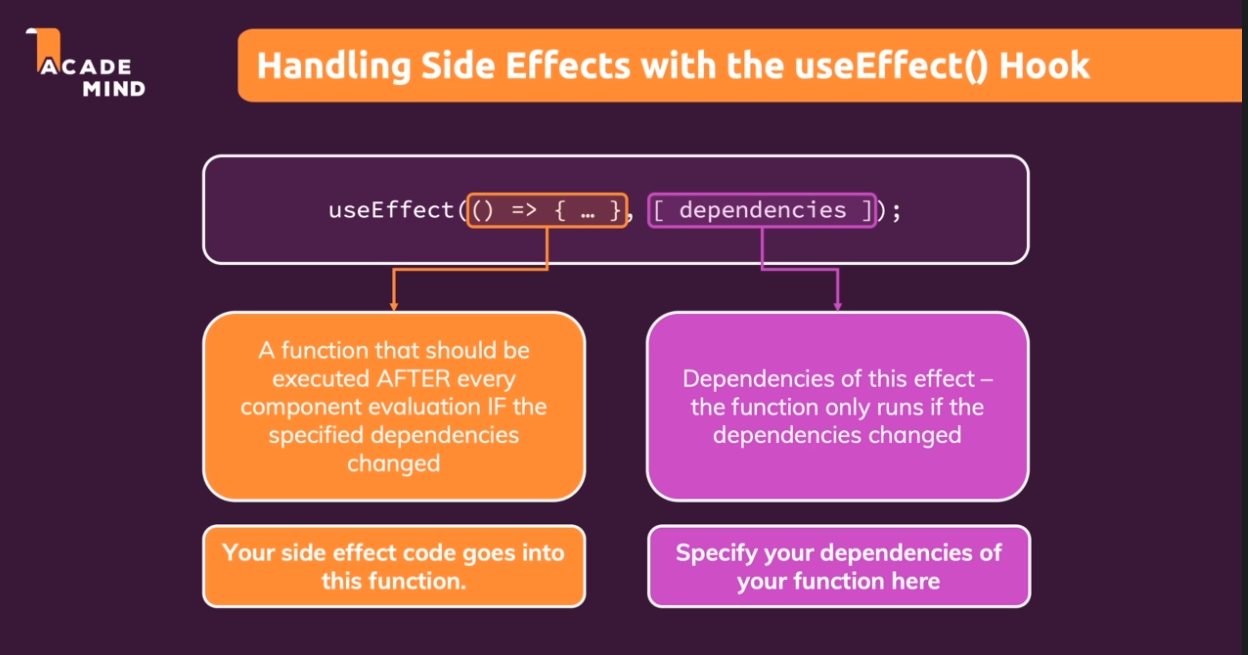
this state changes. That might sometimes be what you want but definitely not always. And if in response to your HTTP request, you for example, eventually change some state, you would even create an infinite loop because



you would send the request whenever the component re-renders and in response to the request you change some state which triggers the function again.

Therefore, such side-effects should not go directly into this App component function because it would most likely create bugs, infinite loops, or simply send too many HTTP requests.

Therefore, we have a better tool for handling side effects, and that’s a special react hook called useEffect(). The useEffect hook is simply another built-in hook, so another function that you can run inside of your component function that will do something special. The useEffect hook is called with two parameters.

The first argument is a function that should be executed AFTER every component evaluation IF the specified dependencies changed. The specified dependencies are the second argument you pass in. That’s an array full of dependencies and whenever such a dependency changes, that first function will re-run. Therefore, in that first function you can put in any side effect code and that code will then only execute when the dependencies specified by you change and not when the component re-renders, only when your dependencies change. 

### 110. Using the useEffect()Hook We have an app where a user has to enter a valid username and at least 7 characters for the password in order to click login. When the user clicks login, they are “logged in”, but if you reload the page, you always lose the login status. That is probably not something you want.

In reality, when you login, a request would be sent to the backend and get back some login data, for example, some token which identifies the user as authenticated. We will cover more on that in the authentication module.

But you want to make sure that this user’s authenticated status is also still there once you reload the page. At the moment we lose this because in the “App.js” file where I manage this isLoggedIn state, it’s just managed as some react state. Therefore, it’s just in the end managed by some JavaScript variable behind the scenes in React. Now, the nature of that is when you reload your application, your entire react script restarts and all variables from the last execution are lost. That’s how the web and scripts and the browser works. This is nothing React specific.

|  |
| --- |
| src/App.js |
| import React, { useState } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {  **const [isLoggedIn, setIsLoggedIn] = useState(false);**  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  setIsLoggedIn(true);  };  const logoutHandler = () => {  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |

Since we lose all the data when this restarts, it would be nice to store it somewhere where it persists the reload. And even better than that, we also want to make sure that whenever this app does start, we check if the data was persisted. And if it is, we log the user in automatically, so that the user doesn’t need to re-enter the details.

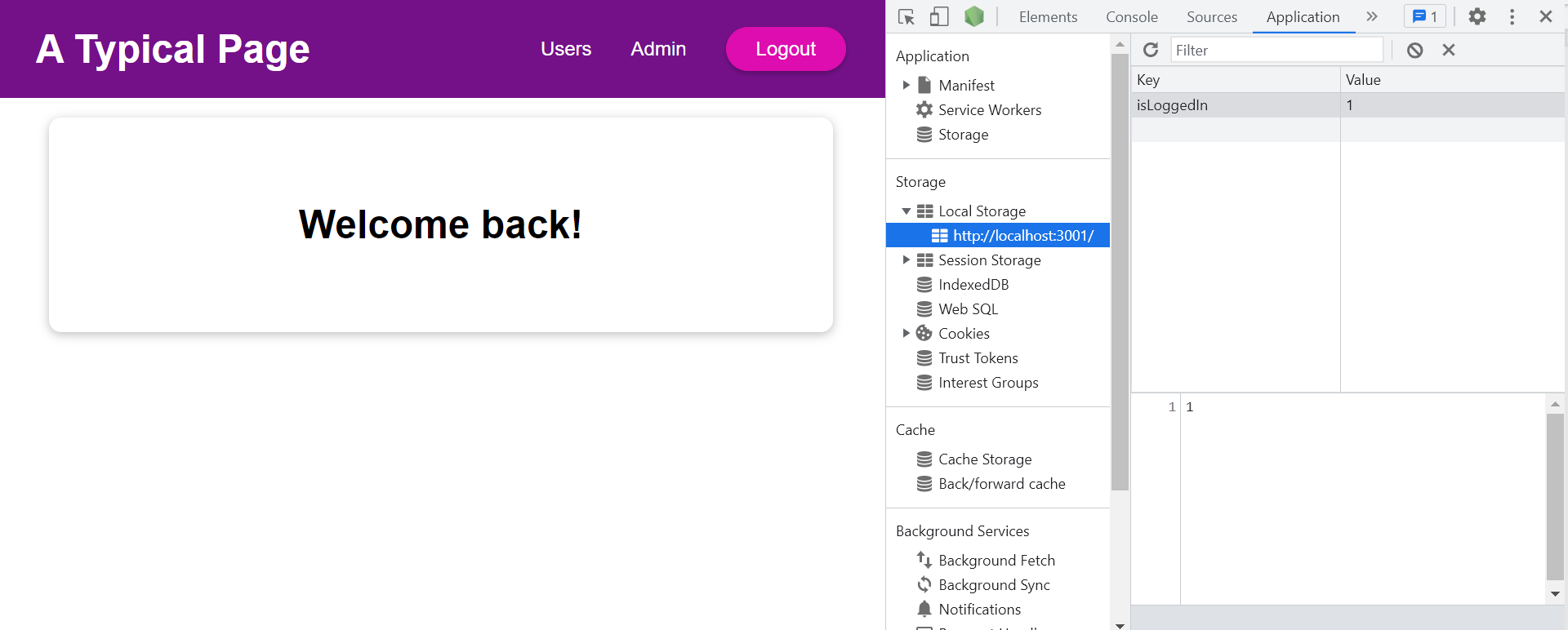
And that’s where we can use useEffect. Now let’s start with storing the data. Here in the loginHandler, I pass true to setIsLoggedIn, which sets isLoggedIn to true. And that’s where I want to store that piece of information in the browser storage. Now the browser has multiple storages that we can use. The most common storage for this would be cookies or local storage, and since it’s particularly easy to work with, we will use local storage. So that’s a storage mechanism built into the browser totally independent of React.

|  |
| --- |
| src/App.js |
| import React, { useState } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {  const [isLoggedIn, setIsLoggedIn] = useState(false);  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  **setIsLoggedIn(true);**  };  const logoutHandler = () => {  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |

Here in the login handler, we could therefore run localStorage. localStorage is a global object that is available in the browser. We can do localStorage.setItem and then give the item any identifier of our choice, like for example, 'isLoggedIn', but that is totally up to you. It should just be a string, though. And the second argument should also be a string, but which you store. And for example, that could be '1' to signal that the user is logged in. '0' could stand for not logged in. Of course, you could also have some identifier like this 'LOGGED\_IN', but this is totally up to you. We will work with '1' and '0'.

|  |
| --- |
| src/App.js |
| import React, { useState } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {  const [isLoggedIn, setIsLoggedIn] = useState(false);  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  **localStorage.setItem('isLoggedIn', '1');**  setIsLoggedIn(true);  };  const logoutHandler = () => {  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |

If I save this and reload everything, if I do login again, we can open up the developer tools -> click the Application tab, there you'll find the Storage section with the Local Storage, if you expand Local Storage, you will find your host, and there you should now see the key value pair, which are the two arguments to localStorage.setItem.



We have this line localStorage.setItem('isLoggedIn', '1'); in the loginHandler because it is a function that executes only when the user clicks the button, which is rare enough and is exactly when we want to store something. This is a use case where we don't need useEffect necessarily.

But how about the scenario when the app restarts because the user left the page and comes back or simply because we reload the page? We then want to check if in local storage we have that key value pair. When the app restarts, the app component function runs again. Now therefore, here of course in the App functional component, we could reach out to localStorage, call getItem, and search for 'isLoggedIn', which will return the items stored there, and we could set this to storedUserLoggedInInformation.

|  |
| --- |
| src/App.js |
| import React, { useState } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {  const [isLoggedIn, setIsLoggedIn] = useState(false);  **const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');**  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  localStorage.setItem('isLoggedIn', '1');  setIsLoggedIn(true);  };  const logoutHandler = () => {  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |

And we could check if the storedUserLoggedInInformation is equal to '1', and if it is equal to '1', we could call setIsLoggedIn and pass in true, so that we set the user to logged in even without the loginHandler function being triggered. And the loginHandler would trigger upon pressing the login button normally just because we know that the user is logged in. We should do this after calling useState. The disadvantage of this approach is that we would create an infinite loop because we check if storedUserLoggedInInformation is stored, if it is stored, we pass true to setIsLoggedIn, which sets isLoggedIn to true. Whenever we call a state-setting function, the entire functional component re-executes. Therefore, the if check would run, would render App again, would find 1, would set it again, and so on.

|  |
| --- |
| src/App.js |
| import React, { useState } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {  const [isLoggedIn, setIsLoggedIn] = useState(false);  const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');  **if (storedUserLoggedInInformation === '1') {**  **setIsLoggedIn(true);**  **}**  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  localStorage.setItem('isLoggedIn', '1');  setIsLoggedIn(true);  };  const logoutHandler = () => {  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |

Therefore, we need useEffect here because that allow us to control when this runs.

|  |
| --- |
| src/App.js |
| import React, { useState } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {  const [isLoggedIn, setIsLoggedIn] = useState(false);  **const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');**  **if (storedUserLoggedInInformation === '1') {**  **setIsLoggedIn(true);**  **}**  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  localStorage.setItem('isLoggedIn', '1');  setIsLoggedIn(true);  };  const logoutHandler = () => {  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |

Hence, from react, besides useState, we import useEffect.

|  |
| --- |
| src/App.js |
| import React, { useState, **useEffect** } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {    const [isLoggedIn, setIsLoggedIn] = useState(false);  const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');  if (storedUserLoggedInInformation === '1') {  setIsLoggedIn(true);  }  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  localStorage.setItem('isLoggedIn', '1');  setIsLoggedIn(true);  };  const logoutHandler = () => {  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |

And then here, we can simply call useEffect();

|  |
| --- |
| src/App.js |
| import React, { useState, useEffect } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {    const [isLoggedIn, setIsLoggedIn] = useState(false);  const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');  if (storedUserLoggedInInformation === '1') {  setIsLoggedIn(true);  }  **useEffect();**  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  localStorage.setItem('isLoggedIn', '1');  setIsLoggedIn(true);  };  const logoutHandler = () => {  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |

And now we pass two arguments – the first argument is a function, for example, here we will use an anonymous arrow function. It doesn't have to be an anonymous arrow function but often you use an anonymous arrow function. The second argument is an array of dependencies.

|  |
| --- |
| src/App.js |
| import React, { useState, useEffect } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {    const [isLoggedIn, setIsLoggedIn] = useState(false);  const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');  if (storedUserLoggedInInformation === '1') {  setIsLoggedIn(true);  }  **useEffect(() => {}, []);**  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  localStorage.setItem('isLoggedIn', '1');  setIsLoggedIn(true);  };  const logoutHandler = () => {  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |

We can move the code that is here:

|  |
| --- |
| src/App.js |
| import React, { useState, useEffect } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {    const [isLoggedIn, setIsLoggedIn] = useState(false);  **const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');**  **if (storedUserLoggedInInformation === '1') {**  **setIsLoggedIn(true);**  **}**  useEffect(() => {}, []);  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  localStorage.setItem('isLoggedIn', '1');  setIsLoggedIn(true);  };  const logoutHandler = () => {  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |

into the anonymous arrow function. We can now run this code in useEffect that we don't want to run directly in the component function.

|  |
| --- |
| src/App.js |
| import React, { useState, useEffect } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {    const [isLoggedIn, setIsLoggedIn] = useState(false);  useEffect(() => {  **const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');**  **if (storedUserLoggedInInformation === '1') {**  **setIsLoggedIn(true);**  **}**  }, []);  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  localStorage.setItem('isLoggedIn', '1');  setIsLoggedIn(true);  };  const logoutHandler = () => {  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |

Because now this function here is executed by react and it is executed after every component re-evaluation. So after the App component runs, this anonymous arrow function in useEffect will run.

|  |
| --- |
| src/App.js |
| import React, { useState, useEffect } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {    const [isLoggedIn, setIsLoggedIn] = useState(false);  useEffect(**() => {**  **const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');**  **if (storedUserLoggedInInformation === '1') {**  **setIsLoggedIn(true);**  **}**  **}**, []);  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  localStorage.setItem('isLoggedIn', '1');  setIsLoggedIn(true);  };  const logoutHandler = () => {  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |

If you then update the state in the anonymous arrow function, the component will run again, but it will only run after a component re-evaluation (component re-render) if

|  |
| --- |
| src/App.js |
| import React, { useState, useEffect } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {    const [isLoggedIn, setIsLoggedIn] = useState(false);  useEffect(() => {  const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');  if (storedUserLoggedInInformation === '1') {  **setIsLoggedIn(true);**  }  }, []);  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  localStorage.setItem('isLoggedIn', '1');  setIsLoggedIn(true);  };  const logoutHandler = () => {  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |

the dependencies here changed. Now when the app starts for the first time, that will be the case. If the App component function runs for the very first time because your app just started, then the dependencies are considered to have changed because you had no dependencies before you could say. But once the app ran for the first time with this setup here ([]), we have no dependencies of course, but they also didn't change compared to the first execution cycle. So therefore,

|  |
| --- |
| src/App.js |
| import React, { useState, useEffect } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {    const [isLoggedIn, setIsLoggedIn] = useState(false);  useEffect(() => {  const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');  if (storedUserLoggedInInformation === '1') {  setIsLoggedIn(true);  }  }, **[]**);  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  localStorage.setItem('isLoggedIn', '1');  setIsLoggedIn(true);  };  const logoutHandler = () => {  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |

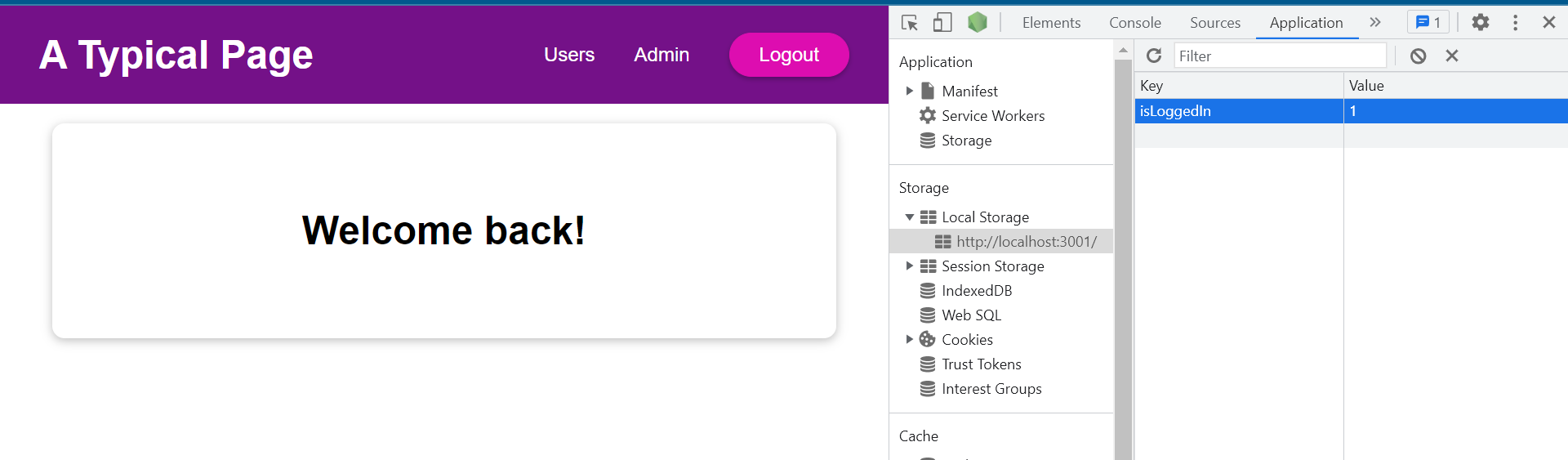
this anonymous function here would indeed only run once, when the app starts, because thereafter the dependencies never change because

|  |
| --- |
| src/App.js |
| import React, { useState, useEffect } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {    const [isLoggedIn, setIsLoggedIn] = useState(false);  useEffect(**() => {**  **const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');**  **if (storedUserLoggedInInformation === '1') {**  **setIsLoggedIn(true);**  **}**  **}**, []);  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  localStorage.setItem('isLoggedIn', '1');  setIsLoggedIn(true);  };  const logoutHandler = () => {  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |

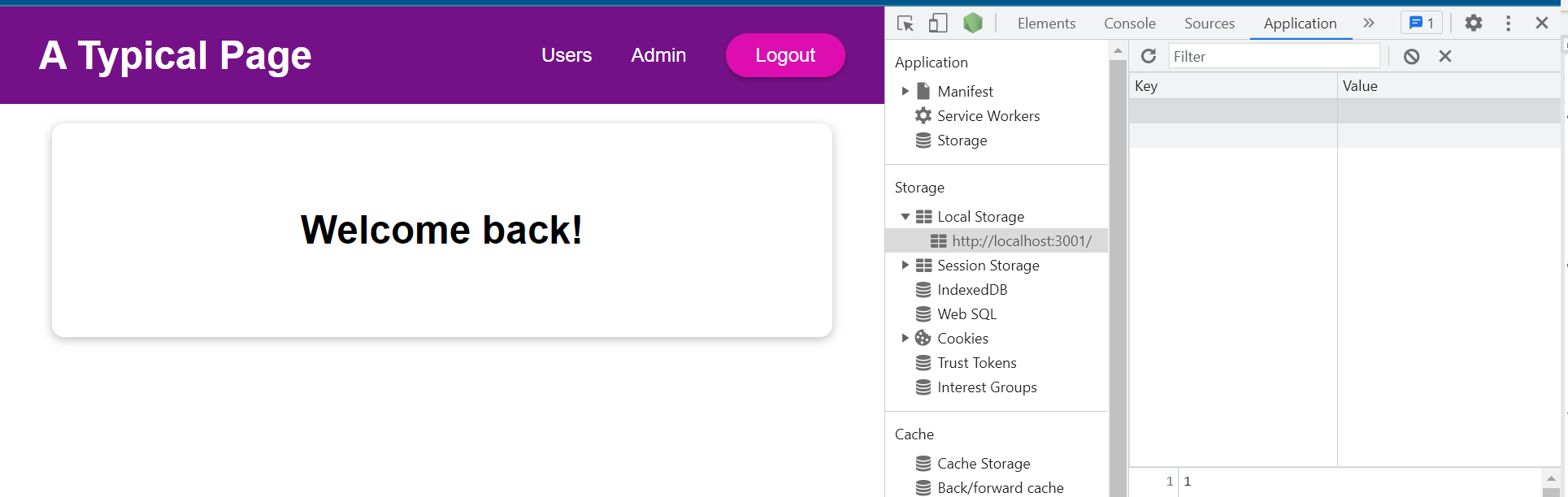
this here specifically has no dependencies.

|  |
| --- |
| src/App.js |
| import React, { useState, useEffect } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {    const [isLoggedIn, setIsLoggedIn] = useState(false);  useEffect(() => {  const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');  if (storedUserLoggedInInformation === '1') {  setIsLoggedIn(true);  }  }, **[]**);  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  localStorage.setItem('isLoggedIn', '1');  setIsLoggedIn(true);  };  const logoutHandler = () => {  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |

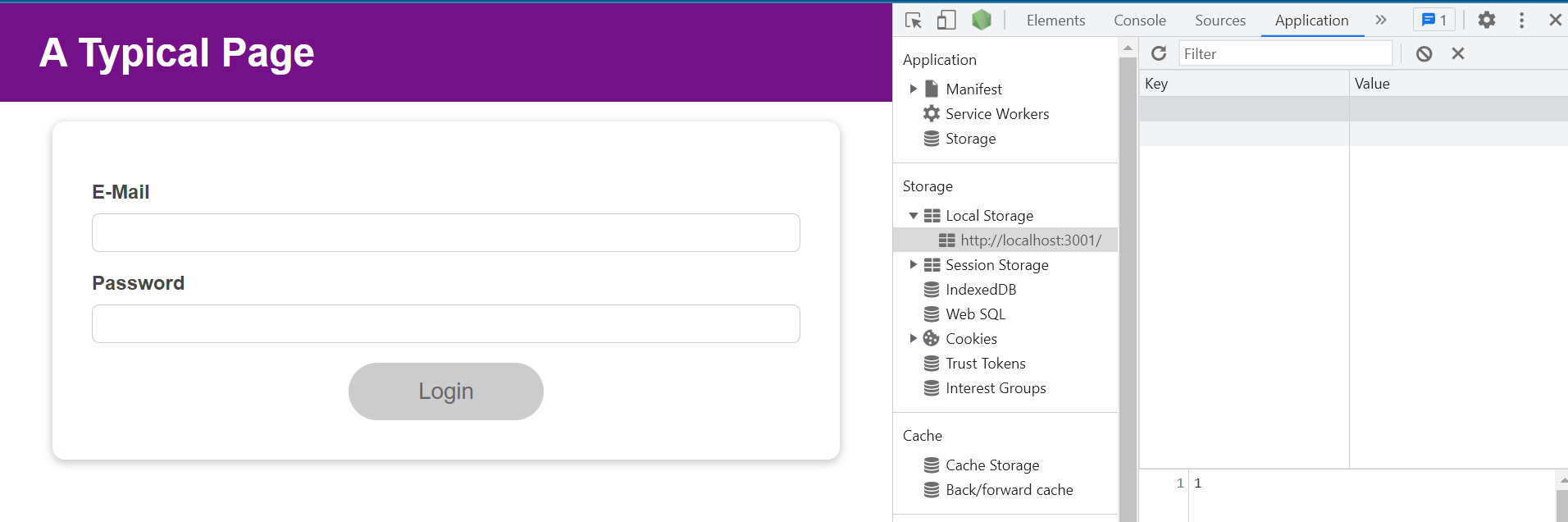
If we save and reload, we should see that we are logged in. We don't end up on the authentication screen but in the logged in state. The reason for that is that we have isLoggedIn stored.



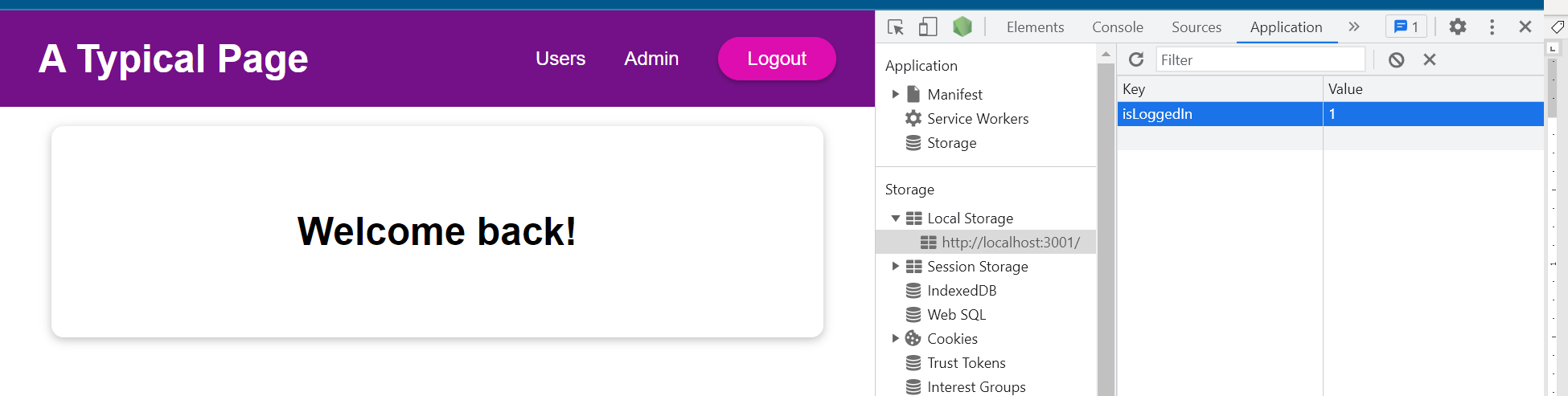
If we were to clear the key and value manually, and



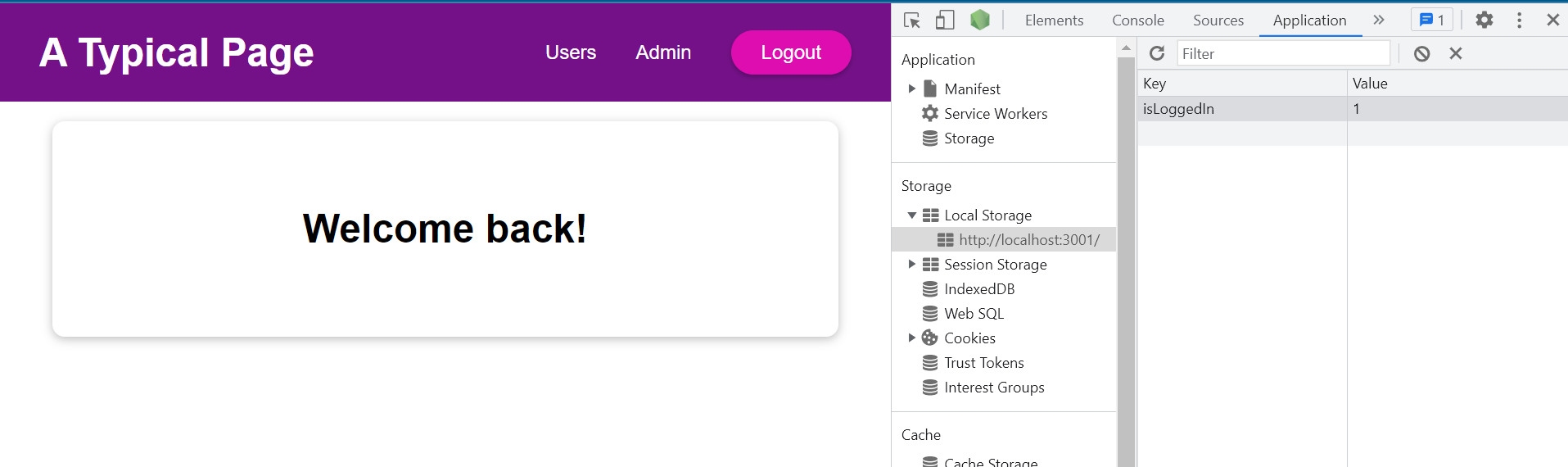
we reloaded the page, we would be back at the login screen.



If we login again, we will see the same behavior as before



If we reload the page, we will stay logged in.



The data fetching is a side effect. It's not directly related to the UI. Of course, the result is, but not the data storage access itself. And we want to run it as a side effect with useEffect in this case to avoid an infinite loop and to make sure that this line of code:

const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');

which could also be performance intensive does not run for every component re-render cycle but only if we wanted to run.

We can refine our data storage example here to make sure that when we click the logout button, we again reach out to localStorage and remove the 'isLoggedIn' key there, so that we don't have to clear it manually.

|  |
| --- |
| src/App.js |
| import React, { useState, useEffect } from 'react';  import Login from './components/Login/Login';  import Home from './components/Home/Home';  import MainHeader from './components/MainHeader/MainHeader';  function App() {    const [isLoggedIn, setIsLoggedIn] = useState(false);  useEffect(() => {  const storedUserLoggedInInformation = localStorage.getItem('isLoggedIn');  if (storedUserLoggedInInformation === '1') {  setIsLoggedIn(true);  }  }, []);  const loginHandler = (email, password) => {  // We should of course check email and password  // But it's just a dummy/ demo anyways  localStorage.setItem('isLoggedIn', '1');  setIsLoggedIn(true);  };  const logoutHandler = () => {  **localStorage.removeItem('isLoggedIn');**  setIsLoggedIn(false);  };  return (  <React.Fragment>  <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />  <main>  {!isLoggedIn && <Login onLogin={loginHandler} />}  {isLoggedIn && <Home onLogout={logoutHandler} />}  </main>  </React.Fragment>  );  }  export default App; |