# React.js Notes – Part 3

Contents

[React.js Notes – Part 3 1](#_Toc126504988)

[Section 9: Diving Deeper: Working with Fragments, Portals, & "Refs" 1](#_Toc126504989)

[99. Module Introduction 1](#_Toc126504990)

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

### 99. Module Introduction

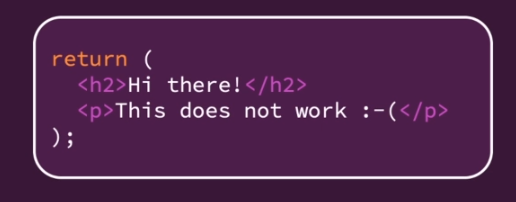
We will look at limitations faced with JSX code and how Fragments can help us overcome those limitations. We'll look at getting a cleaner DOM, better code structure with Portals. We will also look at working with Refs.

### 100. JSX Limitations & Workarounds

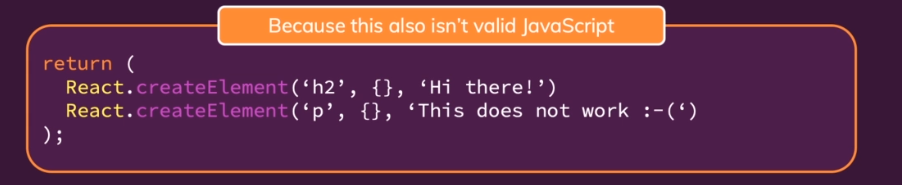
JSX is the code that you return from your components that will in the end be rendered by the real DOM in React.

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.

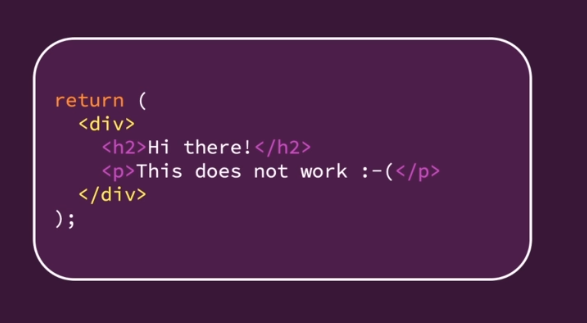


Children elements can be adjacent, but the topmost element can only be one element. This makes sense because in JavaScript we can't return more than one thing.



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.   


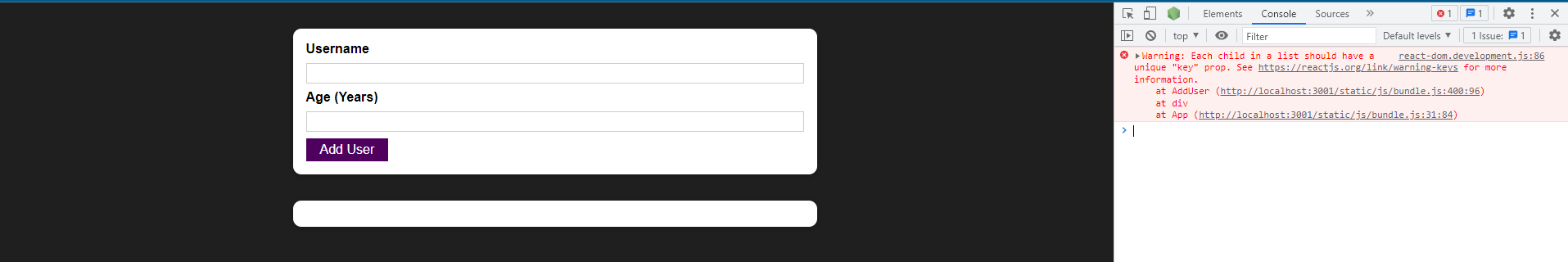
If you wrap the elements in a div, then you are only returning one element. It's similar to returning three numbers by putting them into an array.

**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:



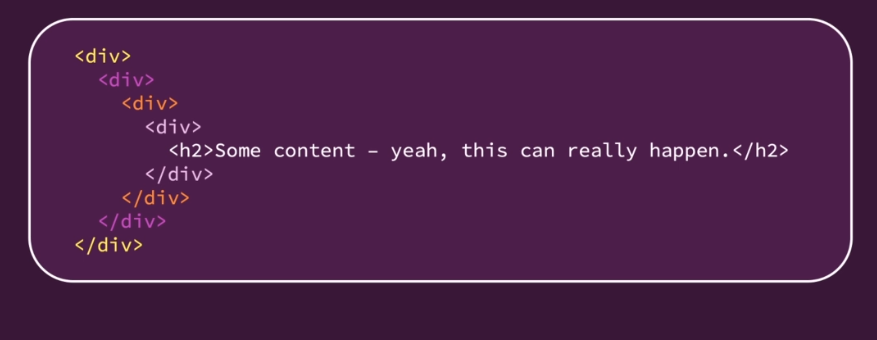
Because we're working with an array of JSX elements, React wants a key on every element. We can simply add a key prop with a custom value to each of the elements.

|  |
| --- |
| 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.