**NOTE 3B: STATE MANAGEMENT AND BEYOND**

**ADDING STATE TO A COMPONENT**

Q: What’s “state” vs “props”?

A:

1. Props refers to attributes from parents components

+) immutable “read-only” data

1. State: mutable data that ultimately affects what is rendered on the page

+) Managed internally by the component itself

+) Meant to change over time, commonly due to user input (ex: clicking on a button on the page)

Q: Example of a state

A:

class User extends React.Component {

state ={

username: ‘Bryan’

}

render(){

return(

<p> Username: {this.state.username}</p>

)

}

}

* Access the state attribute by “this.state.username”
* How the application looks (the app’s UI) is simply a function of the app’s state
* With React, I’m only concerned with 1) what state is in my app, 2) how does my UI change base off of that state?

Q: Git: Untracked files

A: Add them first git add .

For some reason git commit –am doesn’t seem to work the same way

\_We put the State object directly inside the class, not in a constructor() method:

class User extends React.Component {

state = {

username: 'Tyler'

}

}

// rather than

class User extends React.Component {

constructor(props) {

super(props);

this.state = {

username: 'Tyler'

}

}

}

Q: Why do we need to move a variable into State?

A: B/c if not, then React won’t be able to change that state or keep track of the change

So React can know about the change and update the UI

Ex:

class App extends Component {

state ={

contacts : [

{

"id": "ryan",

"name": "Ryan Florence",

"email": "ryan@reacttraining.com",

"avatarURL": "http://localhost:5001/ryan.jpg"

},

{

"id": "michael",

"name": "Michael Jackson",

"email": "michael@reacttraining.com",

"avatarURL": "http://localhost:5001/michael.jpg"

},

{

"id": "tyler",

"name": "Tyler McGinnis",

"email": "tyler@reacttraining.com",

"avatarURL": "http://localhost:5001/tyler.jpg"

}

]

}

render() {

return (

<div>

<ListContacts contacts={this.state.contacts}/>

</div>

)

}

}

* Syntax: state =
* “contacts” , not const contacts
* contacts: , not contacts =

Rule of thumb: If you use a state to render some UI, that state should live in the same component, not outside of it like the variable did to modify it (add, or delete)

\_Do NOT use “props” to initialize state, like this:

this.state = {

user: props.user

}

// cuz even if props are updated, the state won’t change unless the component is refreshed. Also it leads to duplication of data

State:

+) A component’s state can be definied at initialization

+) A component can alter its own internal state

\_React automatically make the necessary updates to the page of state is change

Reconciliation:

\_The process of comparing what changes in the new outputs with the old outputs of the state when updating the page so we don’t have to keep track of exactly what changed

Q: Github—how to change the name of the repo?

A: 1) Go to Github, on main page of the repo, click Settings -> Rename

1. Change the direction of the git remote git remote set-url origin new\_url

**UPDATING STATE:**

\_Can’t update states directly because if so React won’t know what was changed

\_Have to use this.setState

1. ways to do so:
2. Pass a function:

this.setState((prevState) => ({

count: prev.State.count + 1

}));

1. Pass an object, set a completely different state:

this.setState({

username: ‘Jenn’

});

Q: When to use a function vs an object?

A:\_ When the new state of component depends on the previous state => function

Else: object

Q: What do I recognize from this piece of code:

removeContact = (contact) =>{

this.setState((state) => ({

contacts: state.contacts.filter((c) => c.id != contact.id)

}))

}

A: 1)

“removeContact = (contact) => { “ is the same as “function removeContact(contact){ “

2)

I’m actually comparing contact and “c”. Filter here is to say to return a new state without the id of the “contact” variable

\_Then you pass that removeContact function to render:

render() {

return (

<div>

<ListContacts onDeleteContact= {this.removeContact} contacts={this.state.contacts}/>

</div>

)

}

}

\_ In the child component:

Class ListContacts extends Component{

render(){

return(

<ol className= 'contact-list' >

{this.props.contacts.map((contact) =>(

<li key ={contact.id} className='contact-list-item'>

<div className= 'contact-avatar' style={{

backgroundImage: `url(${contact.avatarURL})`

}}/>

<div className='contact-details'>

<p>{contact.name}</p>

<p>{contact.email}</p>

</div>

<button onClick ={() => this.props.onDeleteContact(contact)} className= 'contact-remove'>

Remove

</button>

</li>

))}

</ol>

)

}

}

* That “contact” is the same as the “contact” passed in by this.props.contacts.map((contact) =>

b/c they’re still in the same group

Anyways, you can rename all the “contact” on that child component to something else and it wouldn’t matter because it would just be passed to the parent component automatically

\_Btw, when I pass it back in to the parent component, I don’t have to do onDeleteContact= {this.removeContact()}, I can just do onDeleteContact= {this.removeContact}. In fact, if I put in the (), it won’t compile.

Q: Why?

A: Because if I call (), it will automatically run the method

However, the method itself is just this.removeContact

Q: Why don’t you want to run the method first?

A: Not yet, because I only want to run it when I actually delete the contact.

So if you pass it now, it will just delete the contact

So you pass the method now, without running it

Q: So how does the removeContact method receive the argument?

A: Because the child component ListContacts uses this.props.onDeleteContact(argument) to pass in the argument

Q: So data-binding vs render method. I thought that one way data binding means only the parent component can execute the code, while the children component cannot?

A:

The render method control when something is executed

BUT the data binding controls what gets executed

So, App.JS, which is the parent of ListContact has full control over the data of ListContact. It can directly change the props of ListContact if it wants.

Whereas if ListContact want to change the data of the parent, it has to call the “this” method that the parent actually supplies.

So the parent is still in control. It could pass a different method to ListContact and LC would have to accept it

Q: What’s LifeCycle?

A: it’s an order of operations of how things when they operate under the hood. It always happens in that order:

1. Construct
2. Mount
3. Update props
4. Dismount
5. Etc.

I can plug in to that order by calling any of these methods for things to happen at a particular point in time

\_So to change my state, I should use setState, since it will interrupt the regularly scheduled lifecycle to update my state, whereas just doing this.state ={} will not

\_Every time a local state changes, React will trigger a re-render of the component by calling its render() method

\_ 2 ways to change State with setState:

ex:

class Email extends React.Component {

state = {

subject: '',

message: ''

}

// ...

});

*First way:*

this.setState({

subject: 'Hello! This is a new subject'

})

This way, we can leave this.state.message as-is, but replace this.state.subject with a new value.

*Second way:*

We can use setState() is by passing in a function rather than an object. For example:

this.setState((prevState) => ({

count: prevState.count + 1

}))

Here, the function passed in takes a single prevState argument. When a component's new state depends on the previous state (i.e., we are incrementing count in the previous state by 1), we want to use the functional setState().

* So, state updates can be asynchronous

Q: How?

A: <https://facebook.github.io/react/docs/react-component.html#setstate>

<http://rowanmanning.com/posts/javascript-for-beginners-async/>

RECAP: Use this.setState() to change. Every time the state is changed, React will know and call render() to re-render the component

Q: What does asynchronous mean?

A: \_Synchronous means that JS has to go through all the code in an order to execute them

\_However, async means that JS will put the request on a separate thread, moves on to the next functions in the order, and when it gets its necessary data, then it runs the callback function

Ex:

1. Synchronous:

var article = readSync(article\_loc);

console.log(article);

var authors = readSync(authors\_loc);

console.log(authors)

1. Asynchronous:

readAsnc(article\_loc, function(){

console.log(article);

}

readAsync(authors\_loc, function(){

console.log(authors)

}

Yellow: request

Blue: call back (function)

1. So the Sync version is: it has to execute article, and then authors
2. But Async is like this: It passes the request to the data on a separate thread, then it moves to a second function (readAsync(authors\_loc) . And then when it gets the data, then it execute the callback. So no longer does it have to wait for the first function to execute and then move on to the second function, but it’s non-blocking
3. So if there’s a time (3000 for example, then it’ll time out that time, but it doesn’t just block other functions to be executed during that time, it’ll simply move to the next function. Then when that 3000 time is up, it executes the callback function of the first function

Q: If JS is a single thread language, where does it put the request to?

A: JS that you write runs on 1 thread. But requests such as the async request are passed to a separate thread

Q: this.setState(updater, [callback])

A: updater is the current state. So once the state’s updated, it’ll execute the callback function.

So if this piece of code’s executed, it won’t block other codes from executing

**PROPTYPES:**

Q: What’s PropTypes?

A: A package that lets us define the data type we want to see right from the get-go and warn us during development if the prop that’s passed to the component doesn’t match what is expected

Npm install –save proptypes

So I have to “import PropTypes from ‘prop-types’ “

And then define:

static propTypes ={

contacts: PropTypes**.array.isRequired,**

onDeleteContact: PropTypes.**func.isRequired**

}

Other things: PropTypes.string.isRequired,

PropTypes.number.isRequired

// make it static because it is available everywhere in the component

**CONTROLLED COMPONENTS:**

\_Components which render a form, but the source of truth for that form state lives inside the component state rather than inside the DOM

Q: Why?

A: So React can control it.

Also

1. It supports instant input validation
2. Conditionally disable/ enable buttons
3. It can enforce input formats

Meaning that the UI can easily be updated with user inputs

Ex:

Class NameForm extends React.Component {

state = {

email: ‘ ‘

}

handleChange = (event) => {

this.setState({email: event.target.value})

}

render(){

return(

<form>

<input type = “text value={this.state.email} \>

onChange={this.handleChange} \>

</form>

)

}

}

Q: state = {

query: ''

}

updateQuery = (query1) =>{

this.setState({ query: query1.trim() })

}

What does that do?

1. What’s query.trim?

A: 1) Now the current state has query, which takes in a string.

The updateQuery takes in an argument (querry1), which updates/ sets the value of state.query to query1.trim, which means that it cuts all the spaces around it

Q: Explain this:

render(){

return(

<div className = 'List-contacts'>

{JSON.stringify(this.state)}

<div clasName= 'list-contacts-top'>

<input

className= 'search-contacts'

type= 'text'

placeholder= 'Search contacts'

value={this.state.query}

onChange={(event) => this.updateQuery(event.target.value)}

/>

</div>

….

A 1) JSON.stringify is to convert the JSON data into string. In this case it’ll display whatever you type in (this.state) as a string without blank space. Also {} is the JS mark

1. The default value is this.state.query, but onChange (meaning when someone types something in), it’ll update the Query using event.target.value, meaning the value of whatever I type in will be passed to the updateQuery method
2. Also the value attribute is set on the <input> element. The displayed value will always be the value in the component’s state because the state should be the single source of truth

Q: What’s a Controlled Component?

A: An element whose value React holds the ultimate control of

+) Has to render a form

Q: In this example, how does user input affects the ListContacts component’s own state?

A:

To recap how user input affects the ListContacts component's own state:

1. The user enters text into the input field.
2. An event listener invokes the updateQuery() function on every onChange event.
3. updateQuery() then calls setState(), merging in the new state to update the component's internal state.
4. Because its state has changed, the ListContacts component re-renders.

Q: ‘const’ vs ‘let’. Which is stricter?

A: ‘const’

“Let” can be reassigned

Q: What does this code do?

render(){

let showingContacts

if(this.state.query){

const match = new RegExp(escapeRegExp(this.state.query), 'i')

showingContacts = this.props.contacts.filter((contact) => match.test(contact.name))

} else{

showingContacts = this.props.contacts

}

showingContacts.sort(sortBy('name'))

return()

A: 1) You put the code in the render before the return cuz it’s the UI code

1. RegEx = regular expression. Basically that syntax (in yellow) is to escape the regular expression’s special characters, in this case, case-sensitive. React will use these characters as string literals, rather than RegEx characters
2. Match.test(something) is if you type “something” in and it matches the specific pattern located in this.state.query then it returns true
3. You filter so that the contacts.name matches our specific regular expression
4. .sort(sortBy(‘ name’)): So JS’s native .sort is followed by sortBy(‘property of the object that I want to sort). sortBy is the utility helper that I imported in

Q: What’s regular expression?

A: A sequence of characters that defines the search pattern

**USING DESTRUCTURING:**

Q: What does this code do?

A:

const {contacts, onDeleteContact} = this.props

const { query} = this.state

* You just take the properties from the props and the state so that later you can refer to them directly

Ex:

</div>

<button onClick ={() => onDeleteContact(contact)} className= 'contact-remove'>

Remove

</button>

* This used to be “this.props.onDeleteContact

Q: What does this do?

{showingContacts.length !== contacts.length &&(

<div>Hello</div>

)}

A: && is for JSX’s in line if

So the whole line can be written as:

If(showingContacts.length !== contacts.length){

<div>Hello</div>

}

Q: How about this code?

{showingContacts.length !== contacts.length &&(

<div className='showing-contacts'>

<span>Now showing {showingContacts.length} of {contacts.length} total</span>

</div>

)}

A: contacts.length is actually this.props.contacts.length

showingContacts.length sometimes is shorter cuz it got filter()

Q: <div> vs <span>?

A: div is a block element, span is inline.

This means that to use them semantically, divs should be used to wrap sections of a document, while spans should be used to wrap small portions of text, images, etc.

For example:

<div>This a large main division, with <span>a small bit</span> of spanned text!</div>

Note that it is illegal to place a block level element within an inline element, so:

<div>Some <span>text that <div>I want</div> to mark</span> up</div>

...is illegal.

Q: Walk through what this method does

clearQuery= () =>{

this.setState({ query: ' '})

}

A: It actually:

render(){

…

let showingContacts

if(query){

const match = new RegExp(escapeRegExp(this.state.query), 'i')

showingContacts = this.props.contacts.filter((contact) => match.test(contact.name))

} else{

showingContacts = contacts

}

…

return(

<div className = 'List-contacts'>

…

{showingContacts.length !== contacts.length &&(

<div className='showing-contacts'>

<span>Now showing {showingContacts.length} of {contacts.length} total</span>

<button onClick={this.clearQuery}> Show all</button>

</div>

)}

* onClick: so when users click on here, it resets the query to empty
* then, since query is empty, it’s falsy, and so it shows that showingContacts is actually === contacts