# Day 3 Review

Daily Standup

1. Something you learned yesterday.

2. What you're looking forward to learning today.

3. Blockers. (confusion, questions, etc).

React State

Setting Initial State

constructor(){

this.state = {}

}

Updating State

--this.setState - merges objects (class)

--useState hook - replaces the old state (function)

const [statefulValue,setStatefulValue] = useState(init);

const stateArray = useState(init);

const statefulValue = stateArray[0];

const setStatefulValue = stateArray[1];

Setting state that depends on previous state

this.setState((current)=>{message: current.message + "!"})

this.setState((current)=>{message: current.message + "!"})

this.setState((current)=>{message: current.message + "!"})

What to put in state

-- data that causes changes

-- user input

-- timer

-- changing data from API

Where to put state

-- in a component that's a common parent to every component that needs that state

App

|

Main????

|

ProductList - Cart

Shallow copy of arrays and objects

currentState = [1,2,3]

const newState = [...currentState, 4]

setState(newState)

class components

class React.Component {

constructor(){

this.state = {}

}

setState(objectToMerge){

this.state = {...this.state, ...objectToMerge};

render();

}

}

class MyComponent extends React.Component {

constructor(){

this.state = {

message: 'Welcome'

}

}

translateMessage(lang){

const translation = googleTranslate(this.state.message, lang);

this.setState({message: translation});

}

render(){

{this.state.message}

}

}

Context

const export userPrefs = useContext({theme:'dark',timezone:'PST'})

import {userPrefs} from '';

{userPrefs.theme}

Refs

// a way to work underlying DOM nodes

componentDidMount(){

termsOfUseRef.current.focus()

}

const termsOfUseRef = useRef();

<textarea ref={termsOfUseRef}>

<p>Here are the terms of use</p>

</textarea>

Event

SyntheticEvent

Event listener attributes

onClick

onMouseOver

onSelect

onChange = { handleChange }

Event handler functions

handleChange(e){

setState(e.target.value)

}

Forms

--Controlled input – has value attribute, change by calling a function

<input type="text" value={textFieldValue} onChange={(e)=>handleChange(e.target.value)} />

--Uncontrolled input

<input type="text" id="textInput" ref="textInputRef" />

event.target.value - gives the value of an input

<textarea ref="textRef"></text>

textRef.current.value - access value using a ref

Component Lifecycle - available in Class components

Lifecycle events and methods

* mount / unmount
  + constructor
  + render
  + componentDidMount
    - ajax
  + componentWillUnmount
    - cleanup
* updating
  + componentWillUpdate
  + render
  + componentDidUpdate
* error
  + use for ErrorBoundarys to catch errors in children components

<ErrorBoundary>

<App />

</ErrorBoundary>

function App(){

return (

<DashBoard>

<SearchBox />

<CustomerGrid />

<ErrorBoundary>

<CustomerDetails />

</ErrorBoundary>

</Dashboard>

function App(){

const [customers, setCustomers] = useState([]);

return (

<CustomerList customers = {customers}/>

<CustomersListedByLocation customers = {customers}/>

)