## React Fundamentals

# Patterns & Practices

DevelopIntelligence
A PLURALSIGHT COMPANY

# Topics

- Day 4 Review
- PropTypes
- Create React App + TypeScript
- Higher Order Components
- Introduction to Centralized State with Redux
- useReducer()

Think, Discuss, and Share

# What is the purpose of React hooks?

Think, Discuss, and Share

# Have you heard about React Dev Tools?

## Patterns & Practices

# PropTypes

# PropTypes

- Allows for typechecking of individual props
- Can define default prop values
- Package is automatically included in Create React App projects

```
import PropTypes from 'prop-types'

function SomeChildComponent(props) {

 return (
  <div>
   <div>{props.firstName} {props.lastName}</div>
   <div>{props.age}</div>
   <div>
    Likes Pineapple: {props.likesPineapple ? 'Yes' : 'No’}
   </div>
  </div>
 )
}

// use camelCased 'propTypes' rather than imported `PropTypes' here
SomeChildComponent.propTypes = {
 firstName: PropTypes.string,
 lastName: PropTypes.string,
 age: PropTypes.number,
 likesPineapple: PropTypes.bool
}

export default SomeChildComponent
```

## Patterns & Practices

# Create React App + TypeScript

## Create React App + TypeScript

- TypeScript can be implemented into new or existing projects scaffolded by Create React App
- Uses `.tsx` file extension – TypeScript with JSX – for components

```
npx create-react-app my-app --template typescript
```

```
npm install typescript @types/node @types/react @types/react-dom @types/jest
```

## Patterns & Practices

# Higher Order Components

# Higher Order Components

- A function that accepts a component and returns a new component
- Provides props to wrapped component
- Allows for the reuse of component logic

```
import { useState } from 'react'

const withCounter = WrappedComponent => {
 function WithCounter(props) {
  const [count, setCount] = useState(0)

  const incrementCount = () => setCount(count + 1)

  const decrementCount = () => setCount(count - 1)

  return (
   <WrappedComponent
    count={count}
    incrementCount={incrementCount}
    decrementCount={decrementCount}
    {...props}
   />
  )
 }

 return WithCounter
}

function SomeCountingComponent(props) {
 return (
  <div>
   <h1>{props.count}</h1>
   <button onClick={props.incrementCount}>Increment</button>
   <button onClick={props.decrementCount}>Decrement</button>
  </div>
 )
}

export default withCounter(SomeCountingComponents)
```

## Patterns & Practices

# Centralized State with Redux

# Centralized State with Redux – Core Concepts

- **Store** – Object that holds the state
- **Action** – Object that describes the change. Must include a ”type” field
- **Reducer** – Function that accepts the current state and an action, and returns the updated state

## Patterns & Practices

# useReducer

# useReducer()

- Accepts a reducer function and returns current state and a dispatch method (like Redux!)
- Useful for updating state with complex structure / multiple sub-values, or when the next state depends on the previous state

```
import { useState, useReducer } from 'react';

function Counter() {
 const [input, setInput] = useState(0)
 //first arg, reducer function
 const [count, dispatch] = useReducer((state, action) => {
  switch(action.type) {
   case 'increment':
    return state + (action.payload || 1)
   case 'decrement':
    return state - (action.payload || 1)
   default:
    return state
  }
 //second arg, initial value
 }, 0)

 return (
  <div>
   <h1>Count: {count}</h1>

   <button onClick={() => dispatch({ type: 'increment', payload: Number(input)
})}>Increment</button>

   <button onClick={() => dispatch({ type: 'decrement', payload: Number(input)
})}>Decrement</button>
  </div>
 )
}

export default Counter
```

# Let’s try it!

DevelopIntelligence
A PLURALSIGHT COMPANY