REACT - REDUX + ES6 JS / JSX - BABEL

Export / Import js

file person.js

const person = {
name: "Ale"
}

export default person → the (default) export obj / entity in a page

file utility.js

export const clean = () =>
export const baseData = 10

In IMPORT case with default we don't need to report the exact name of what we need to import

app.js

import prs from './person.js' / import {baseData} from
" person from './person.js' / import {clean} from './utility'

{ clean as Alias } import { clean, baseData } from './utility.js'

import * as bundle from './utility.js'

Classes

Create/inherit with extends other class

can skip constructor () {
myProp = "value"

Spread and REST

→ array into elements (...array)
objs into his prop.

Rest fn (...arg)
return arg.sort()
returns an array (sorted)

Clone obj or Array

Spread OPERATOR "..."

```
const person = {
    name = "max" }

const secondPerson = {
    ...person
}
```

-> new objs and NOT a reference

REACT - new Project

Create a react work compatible env for All the browsers

1. manage Dependency npm
2. bundler Webpack
3. Compiler Babel + Presets
4. Developer Server web server

use: Create react App

- how JSX is compiled
- using state for managing attributes, "props" inside only (valid extending a class & component with component

REACT

class app extends Component

use this.setState (new version of We want to change)

DO NOT: this.state.person[0].name = ""

REACT HOOKS > 16.8 can be called multiple states

- useState() -> returns Array with current state function to update the state

const [current, updateState] = useState() -> REPLACE the state

> destructuring

no need to use {Component} and extends in a class

import React, {useState} from 'react' -> use in a function like in person ...

Reply

→ PRESENTATIONAL → view components

stateless vs statefull
→ dont use states in all of the form → use state

pass methods with props

const handler=(new name) => { ... }

<person
click = {this.handler}
can use bind(this, 'new name') to bind this and new arguments

() => this.handler('Abe') → can rerender the page often

condition : INSIDE RETURN
only ternary comparison
{
condition ?
<div>
</div> : null
}

OR

person = null outside return
if(condision){
person = (
<div>
</div>
);

RETURN (
{person
});

WORKING WITH STYLES

USING RADIUM ← High order component
to wrap component with new functionality

! NOT TO USE WITH RADIUM

Import <style Root> from 'radium'
for using query media selector

CONST StiledDIV= styled.div` style `

67 can us styled-components

styled.div` style ` // return a JSX component with an class name
use it as a JSX comp. RET.

• styled-components

```
const comp = styled.div`
  background-color: ${ condition ? "green" : "RED" };
`
```

--- USE ONE ONLY ---

• Radium

If you need @media queries in the main application will need to import { StyleRoot } 'radium' and wrap the the <App> with <StyleRoot> on inject the style with JS ~~style~~ objects.

```
':hover': {
  ...
}
```

```
'@media (min-width: 500px)': {
  { ... }
}
```

CSS Modules → only on ~~so~~ React Scripts > 2.0

import classes from "./Person.module.css";

with < 2 you need to enable the with the eject

config the setting >

Error and Debugging

React and source maps for DEV tools in Chrome

• Error Boundry

- Class that extends Components that can catch Errors and Display Custom Messages.

HOC → wrap the component that may fail and you can controll that

<ErrorBoundry> <Person /> </ErrorBoundry>

passes the key to the higher component.

METHOD:

componentDidCatch ← (error, info) | can use a state to track the error