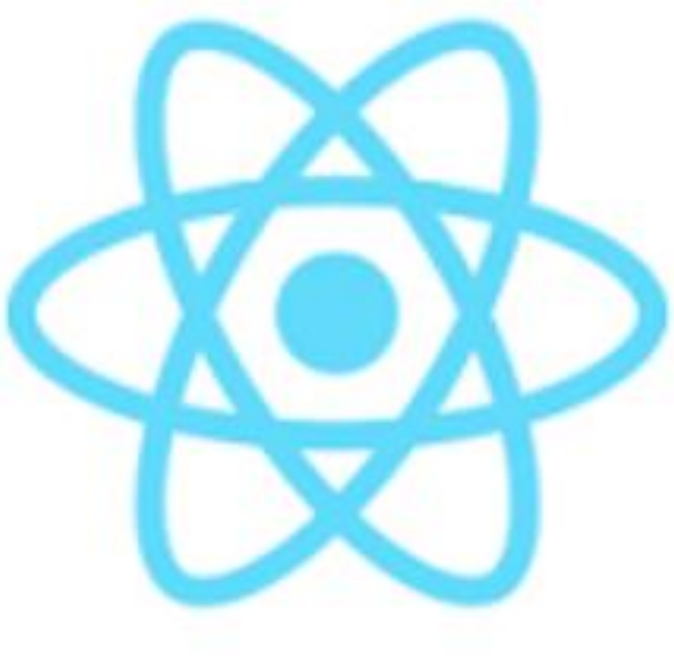
Facebook React + Material-UI

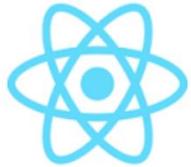


A library for building user interfaces

CENTRIQ TRAINING

Alumni Continuing Education
10/2/2019

Doug Hoff

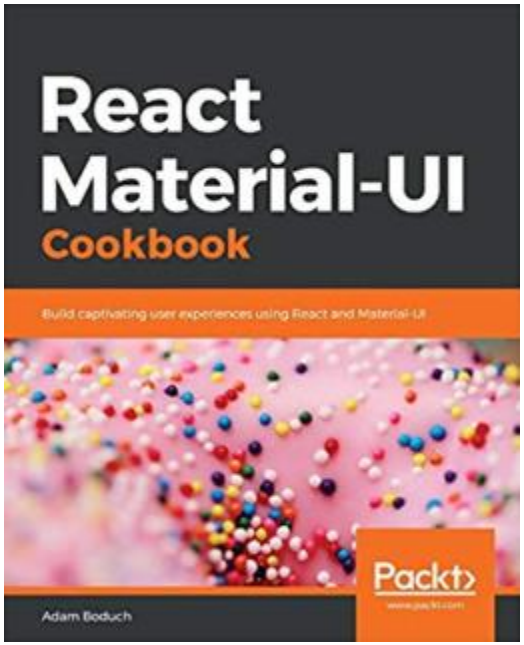


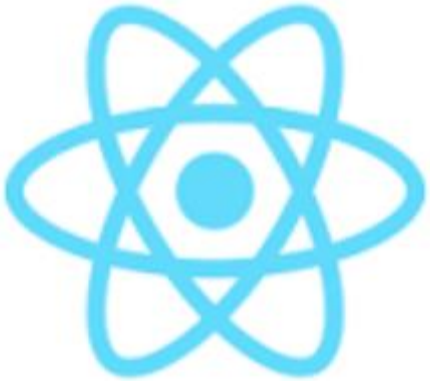
- Centriq instructor
  - Since 2005
  - Programming – Java, JavaScript, C#
  - Business – ITIL, business analysis, software testing
  - http://doughoff.com
  - @doug_hoff

https://www.amazon.com/React-Material-UI-Cookbook-captivating-experiences/dp/1789615224/ref=sr_1_3

Books

- React Material-UI Cookbook: Build captivating user experiences using React and Material-UI by Adam Boduch
- March 2019



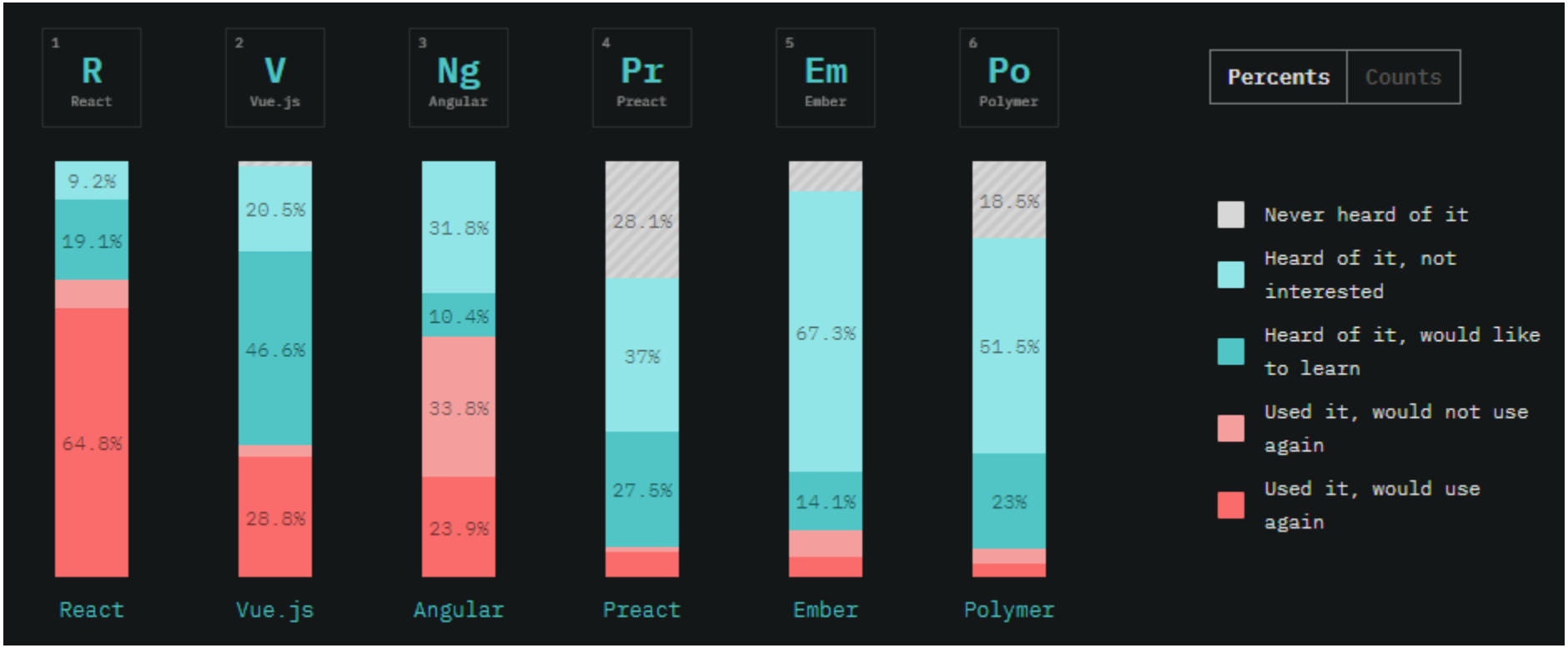


React basics

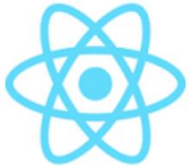
https://2018.stateofjs.com/front-end-frameworks/overview/

Front-end frameworks

- React is the highest rated JS web framework

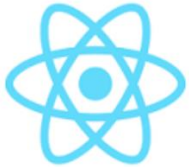


How React renders the page



- A (virtual) copy of the DOM is stored in memory
- Updates are very fast (no rendering)
- Rendering workflow
  - **diffing** - comparing snapshot of browser DOM with new virtual DOM to figure out what's changed.
  - **patching** - executing only the necessary DOM operations to make changes
  - **rendering** – changed DOM is redrawn

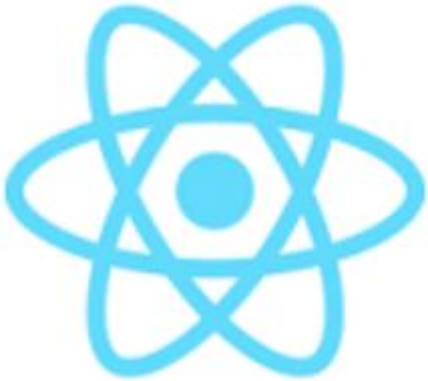
https://reactjs.org/



History

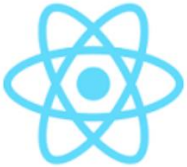
- 16
  - 16.9.0 (August 8, 2019)
  - 16.8.1 (February 6, 2019)
    - added Hooks
  - 16.0.0 (September 26, 2017)
- 15.0.0 (April 7, 2016)
- 0.14.8 (March 29, 2016)
- 0.3.0 (May 29, 2013) – initial release

```
const element = <h2>Running
{React.version}</h2>;
```



Install

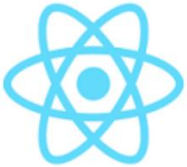
https://facebook.github.io/create-react-app/



create-react-app

- Best way to create dev environment
- Docs
  - https://reactjs.org/blog/2018/10/01/create-react-app-v2.html (Oct 2018)

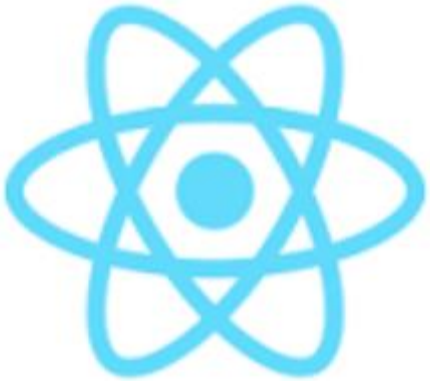
https://codepen.io/doughoff/pen/VJmxBE



CodePen setup

- **React template** pen in my account
- Uses Babel pre-processor and unpkg imports
- Uses special import for Material-UI

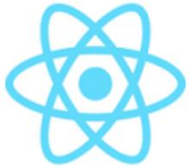
```
const { } = MaterialUI;

const element = 'text/html/JSX to render' ;

ReactDOM.render(element,
document.querySelector("#root"));
```



Pseudo-HTML DSL

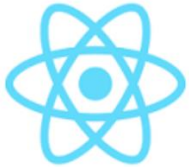
JSX

JSX basics



- Not required for React
- Extended JS syntax, not HTML
  - looks like strings embedded in JS without quotes
  - can contain JS expressions
  - turned into HTML by renderer
- Used for any render target
  - React Web
  - React Native
  - React Desktop

https://codepen.io/doughoff

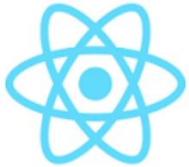


JSX exercises

- Use the **React template** pen
- Clear console.
  - lower left corner – CodePen is OK, use DevTools instead
- Leave
  - first line (Material-UI import)
  - last line (ReactDOM.render…)
- Replace middle with example code and wait.

```
const element = <h1>Hello, world!</h1>;
```

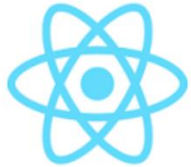
Rendering



- Use a constant variable to store JSX
- Select DOM element to render to

```
const element = <h1>Hello, world!</h1>

ReactDOM.render(element,
  document.getElementById('root')
);
```

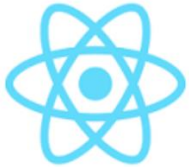
Rendering



- Alternative forms of triggering a render
  - querySelector( ) vs. getElementById( )
  - JSX passed as an argument

```
const element = <h1>Hello, world!</h1>
ReactDOM.render(element,
document.querySelector("#root"));
  // or
ReactDOM.render(<h1>Hello, world!</h1>,
  document.querySelector("#root")
);
```

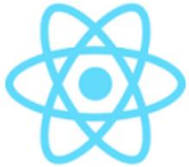
JSX is like HTML



- Use any html element
  - one root element, must have end tag
  - tag names must be in lower case, PascalCase is for React elements
- Nesting is OK, optionally group lines with parentheses

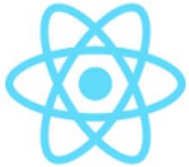
```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>We have <b>amazing</b> products!</h2>
  </div>
);
```

JSX vs no JSX



```
const element =
  <h1 className="greeting">
    Hello, JSX!
  </h1>
;

const element = React.createElement(
  'h1', {className: 'greeting'},
  'Hello, JavaScript!'
);
```

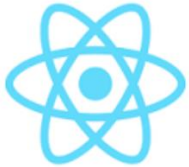
Fragments



- Eliminates rendering a root element
- Wrap your code to render in a <Fragment> element – requires import
  - Removed when rendered
- Easier to use JSX shortcut: <> and </>

```
const element = <>
  <p>First nested element</p>
  <p>Second nested element</p>
  <p>Third nested element</p>
</>;
```

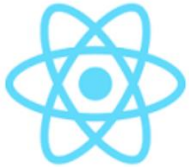
Fragment import



- Seen with destructuring import statement
  - `import React, { Component, Fragment } from "react";`
  - **CodePen**: use <React.Fragment>

```
const element =
<React.Fragment>
      <span>Whoa, </span>
      <span>yeah.</span>
</React.Fragment>
;
```

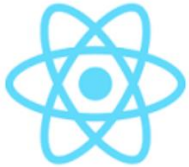
Expression containers



- Curly braces evaluate expressions inside JSX

```
const name = 'Jordan Walke';
const element = <>
  <h2>{ "literal" }</h2>
  <h2>{" "}</h2>
  <h2>{ 2 + 2 } </h2>
  <h2>{ "author: " + name } </h2>
  <h2>{ name.toUpperCase( ) } </h2>
  <h2>{ new Date().toLocaleDateString() } </h2>
</>;
```

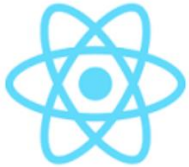
Expression containers with variables



- Variables can be evaluated from
  - strings, numbers, arrays, object fields

```
const company = 'Centriq';
// const company = ["The  ", "Factory"];
const element = <h1>Welcome to {company}</h1>;
-------------------------------------
const time =
 <span>{new Date().toLocaleTimeString()}</span>
const element = <h1>Time is now {time}</h1>;
```

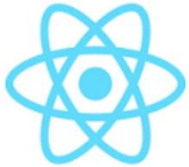
JSX as return values



- Use any function return values of JSX

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}
function getJSXGreeting (user) {
    return <h1>Howdy, {formatName(user ||
{lastName: "stranger"}) }!</h1>;
}
const element = getJSXGreeting(
  //   {firstName:"Doug", lastName:"Hoff"}
);
```

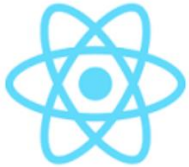
JSX - attributes



- Use JS camelCase property naming
  - not HTML attribute naming
  - className instead of class
- String variables don't need to be quoted again
  - literals do

```
const element = <h2 className='red'>I'm red.</h2>
const element = <h2 tabIndex={0}>I'm first.</h2>
const react = {logo:
'https://cdn4.iconfinder.com/data/icons/logos-
3/600/React.js_logo-512.png'};
const element = <img src={react.logo}></img>;
```

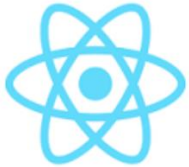
Basic component syntax - function



- Element name is function name
- Function returns what is rendered.

```
function Basic(){
  return 'Basic function component';
}

const element = <Basic/>;
```

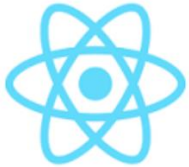
Basic component syntax - lambda



- Element name is lambda name
- Lambda returns what is rendered.

```
const Basic = () => {
  // other statements
  return 'Basic lambda component';
}
const Basic2 = () => 'Basic lambda component';

const element = <Basic/>;
```

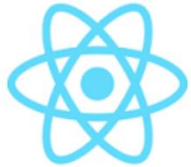
Basic component syntax - class



- Class extends Component
- Render function returns what is rendered.

```
class Basic extends React.Component {
  render()
  {
    return 'Basic class component';
  }
}

const element = <Basic/>;
```

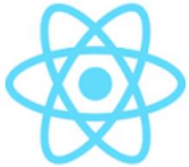
Components as JSX



- Can be standard HTML or user-defined JSX type HTML (component)

```
function Welcome(props) {
  return <h1>Hello, {props.name}.</h1>;
}

const element = <Welcome name="Kitty" />;
```

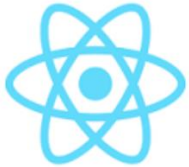
Map objects to elements



- Use Object.keys(someObject) to get an array of keys
- Use non-dot syntax to access values

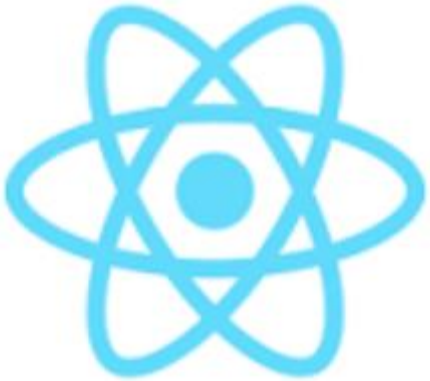
```
const object = {a:'abc', b:2 , c:1.23};

const element = <ul>
  { Object.keys(object).map(key =>
  <li id={key} >
    {key} = {object[key]}
  </li> )}
</ul>;
```

Element lists need keys



- Lists need unique keys for performance
  - React will warn you with an error
  - key attributes do not render
- Using indexes as keys is not recommended
  - ok with no static list or filtering, reordering, or no ids

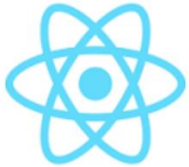
```
const object = {a:'def', b:22 , c:4.56};
const element = <ul>
  { Object.keys(object).map(key =>
    <li id={key} key={key}>
      {object[key]}
    </li> )} </ul>;
```



Properties from the JSX attributes

Props

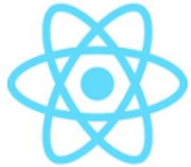
{props.<propertyName>}



- Pass JSX attributes to component properties in the
  - for functions: first parameter
  - for classes: the class

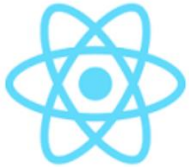
```
const HelloWorld = (props) =>
<h2>Hello, {props.nameFirst}!</h2>;

const element = <HelloWorld
nameFirst='world'/>;
```

Props data



- Short for properties object
  - object fields from attributes in an element
  - first argument passed in by React if function
- Props data should only be updated by another component.
  - changing a property value will not re-render JSX
  - changing the whole props will
- Don't rely on data from a prop
  - unless it's a seed or
  - a single source of truth

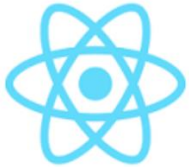
Default properties



- Object defined as a field of the component
- Place after a lambda function declaration
  - hoisting only works on simple function declaration

```
const Greetings = (props) =>
<h3>{props.greeting} {props.firstName}
{props.lastName}!</h3>;

Greetings.defaultProps = {firstName:'John',
lastName:'Smith', greeting: 'Hello,'};

const element = <Greetings firstName='Alonzo'
lastName='Church'/>
```

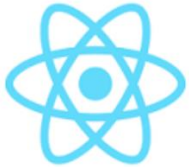
Destructuring props



- Replace props parameter with object declaring props field names in any order

```
const Greetings = ({firstName, lastName,
greeting}) =>
<h2>{greeting} {firstName} {lastName}!</h2>;

Greetings.defaultProps = {firstName:'John',
lastName:'Smith', greeting: "Thanks,"};

const element = <Greetings firstName="Alonzo"
lastName="Church"/>
```

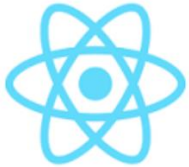
Destructuring for default props



- Values assigned to object will overwrite
  - default values for function components

```
const Greetings = (props) => {
  const {greeting='Ola!', firstName='John',
lastName='Smith'} = props;
  return <h2>{greeting} {firstName}
{lastName}!</h2>;
};

const element = <Greetings firstName="Alonzo"
lastName="Church"/>
```

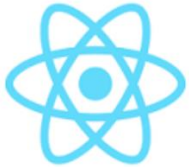
Destructuring and renaming



- Use a colon after the expected variable name to rename it
- Default values follow the new name

```
const Greetings = (props) => {
  const {greeting : g = 'Hey!', firstName : fn,
lastName :ln} = props;
  return <h2>{g} {fn} {ln}!</h2>;
};

const element = <Greetings firstName="Alonzo"
lastName="Church"/>
```

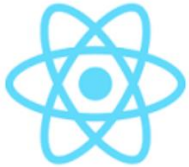
Destructuring nested objects



- Nested object names can also be destructured

```
const alonzo = {firstName:'Alonzo',
lastName:'Church', number: 1234};


const Greetings = ({person, person: {firstName,
lastName, number}}) =>
<h2>Welcome back {firstName} {lastName}!
<br/> Member number={number}<br/>
{person.toString()}
</h2>;


const element = <Greetings person={alonzo}/>
```

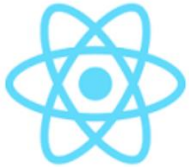
Destructuring with ...restProps



- the spread op groups attributes to pass through

```
const Greetings = (props) => {
  const {greeting='Hi!', firstName='John',
lastName='Smith', ...restProps} = props;

  return <h2 {...restProps}>{greeting}
{firstName} {lastName}!</h2>;
};

const element = <Greetings firstName="Alonzo"
lastName="Church" style={{color:'green',
fontSize:'500%'}} id='greeting' title='hey!'/>
```

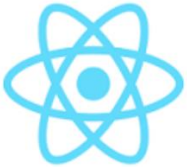
props.children



- props.children passes the body of the component element back

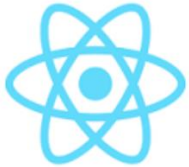
```
const DataColumn = ({db, field, type,
children}) =>
<h2>{field} of {type} {db}: {children} </h2>;

const author = {company: 'Acme'};

const element =
<DataColumn db='Company' field='Name'
type='Education'>{author.company}</DataColumn>
```

Prop drilling is bad



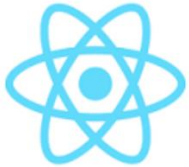
- Passing a prop to a component just so it can pass it to its child is a bad practice
  - hides the child
- A delegation for coupled components
  - Usually this is thought of as a convenient practice
  - as in a constructor in an inherited class delegating the data to its superclasses.
- Context API provides a way to pass data through the component tree without having to pass props down manually at every level.

Prop drilling is bad



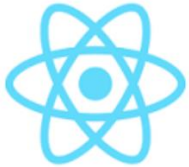
```
const Child = ({className}) =>
<q className={className}>Child controlled by
parent.</q>;

const Parent = ({parentClass, childClass}) =>
<h2 className={parentClass}>
        <Child className={childClass} />
</h2>;

const element = <Parent parentClass={'parent-
class'} childClass ={'child-class'} />;
```

Use composition to cure prop drilling



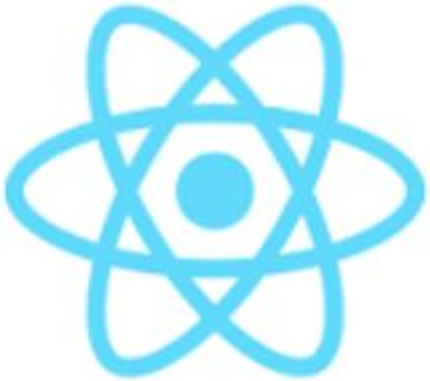
```
const Child = ({childClass}) =>
  <q className={childClass}>
    Child class is {childClass} </q>;

const Parent = ({parentClass, children}) =>
  <h2 className={parentClass}>
    Parent class is {parentClass}<br/>
    {children} </h2>;

const element =
<Parent parentClass= 'parent-class'>
  <Child childClass='child-class'/>
</Parent>;
```

Update by re-rendering



- Use more than one render statement to control rendering

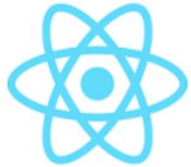
```
function tick() {
  const element = <h2>It is {new
Date().toLocaleTimeString()}.</h2>;
  ReactDOM.render(element, document.
querySelector("#root"));
}

setInterval(tick, 1000);
const element = <h2>Gimme a sec here...</h2>;
```



designing without changing state

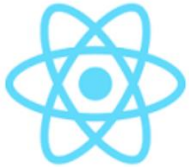
Function components

Component architecture



- Function components are for view content only
  - not business logic
  - not to do database access
  - not a utility function
- Data input can be props or state
  - Forms should use updatable state object in class components or use updatable hooks

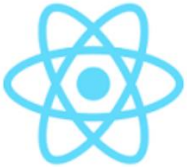
Function components



- How to design
  - split UI into independent, reusable pieces
  - no prop drilling
  - think about each piece in isolation
- Return JSX

```
function Welcome() {
  return (
    <h1>Hello!</h1>
  );}
const element = <Welcome/>
```

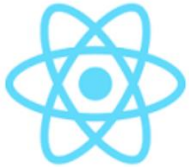
Functions as lambdas



- Lambdas are nice.

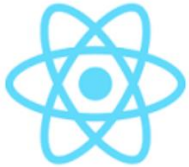
```
const Welcome1 = () =>  <h1>Hello, lambda
1!</h1> ;
const Welcome2 = () => (<h1>Hello, lambda
2!</h1>) ;
const Welcome3 = () => {
  let i = 3;
  return <h1>Hello, lambda {i}!</h1> ;
};

const element = <> <Welcome1 /> <Welcome2 />
<Welcome3 /> </>;
```

Functions are better than classes



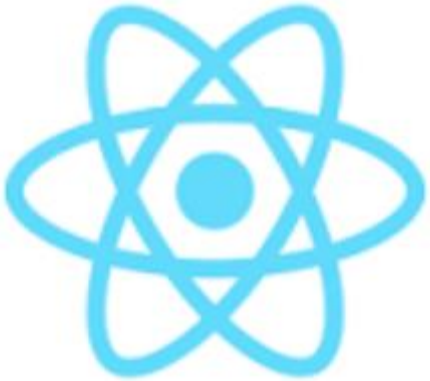
- Code is more simple, reusable, and modular.
  - Logic and presentation separation
  - Easier to test
- Prevents abuse of the setState() API
- Encourages "smart" vs. "dumb" component pattern.
- Allows React to make performance optimizations

Lambdas with props



- Upgrading to lambdas works well
- When using one parameter, the parens are optional… (props) or props

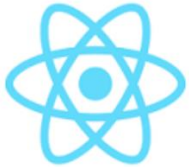
```
const Greetings = (props) =>
<h2>Hey! {props.firstName} {props.lastName}!
</h2>;

const element = <Greetings firstName="Alonzo"
lastName="Church"/>
```



CSS, style, themes

Style

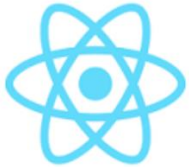
className



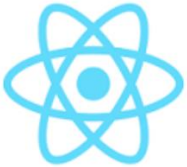
- use the JS property className in JSX
  - translates to the HTML attribute of class

```
.title {color:red}
-----------------------------------
const classes = {title:'title'};
const element = <p
className={classes.title}>Title class</p>;
```

Style attribute



```
const element= <>
<div style= {{
      margin: 50,
      padding: 10,
      width: 300,
      border: "1px solid black",
      backgroundColor: "black",
      color: "white"
}} >Lorem ipsum.</div>
</>;
```



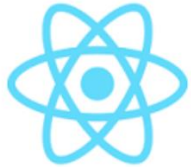
Styles in external file

- Create a special style object that will contain all styles.
  - good practice to place the styles in a separate file

```
const styles = { form: { margin: 50, padding: 10,
width: 300, border: "1px solid black",
backgroundColor: "black", color: "white" },
inputGroup: { marginBottom: 10 }, input: {
backgroundColor: "#EFEFFF", marginLeft: 10 }, error:
{ color: "red", margin: 5 }};
```

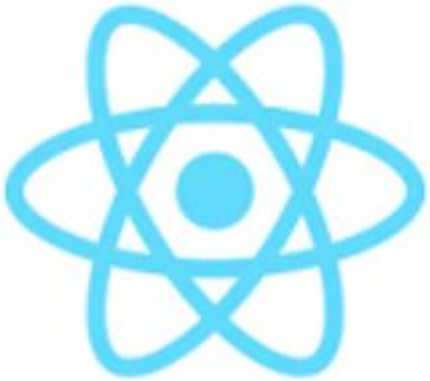
```
import styles from '../style';
return <><div style={styles.form}>Lorem ipsum</div>
<div style={styles.inputGroup}></>;
```

CSS-in-JS



- https://material-ui.com/css-in-js/basics/
- JSS - https://cssinjs.org
  - the core of Material-UI's styling
- Many advantages over LESS, SASS, CSS Modules, etc.
- for large scale CSS

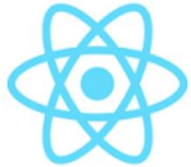
```
npm install @material-ui/styles
```



When you need state

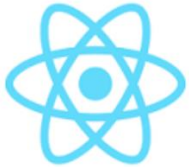
Class components with state

Properties vs. state



- Both provide data for components
- State
  - mutable during component's lifecycle
  - scoped by component
- Props
  - read-only
  - tend to be coded as function components
- Together?
  - Updated asynchronously so don't use props to update state.

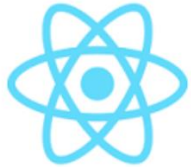
State initialization – class property



- state is a built-in property, aka field
- Use an object (not an array)

```
class Greeting extends React.Component {
  state = { one: 'Hello', two: 'whoever' };
  render(){
    const { one, two } = this.state;
    const { name } = this.props;
    return <h1>{one}, {name || two}!</h1>;
  }
}
const element = <Greeting name='React' />;
```

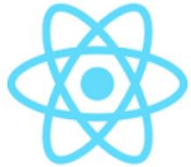
State initialization - constructor



- Initializing state inside the constructor is an anti-pattern?
  - Unnecessary boilerplate
  - State open to mutations.
- Ok for a starting value though.

```
constructor(props) {
  super(props);
  this.state = {date: new Date()};
}
```

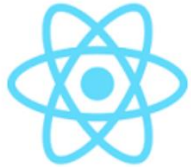
State initialization – no constructor



- constructor not necessary
- class property is new
- better choice in general
- update in exercises if you want

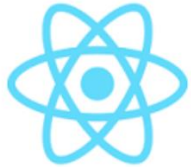
```
constructor(props) {
  super(props);
}
state = {date: new Date()};
```

Updating state with setState( )



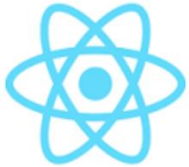
- Classes use setState( )
  - this.setState({itemToUpdate: newValue})
- Updater form is recommended.
  - argument is a function not an object
  - https://reactjs.org/docs/react-component.html#setstate
- Functions must use Hooks with useState( )
- Re-render occurs after update.

Updating state



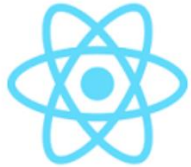
- Setting new state by an object is asynchronous
- Multiple changes will not show intermediate updates and go straight to final rendered result.

State update with setState(object)



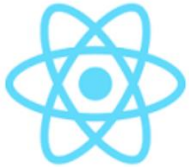
```
class Testing extends React.Component {
  state = { counter: 1 };
  constructor(props) {
    super(props);
    setTimeout(() => this.tick(), 1000);
    setTimeout(() => this.tick(), 2000);
  }
  tick() {
    this.setState({ counter: this.state.counter + 1
});
  }
  render() {
    return <h1>Testing, {this.state.counter}!</h1>;
  }
}
const element = <Testing />;
```

State update by setState(f)



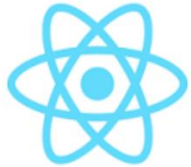
- Updater function, usually a lambda, will make the change
  - recommended by Facebook
- First parameter, prevState, is for current state
- Returns an object, in the same structure as the state object, to modify new values
  - not always a complete object

State update by setState(f)



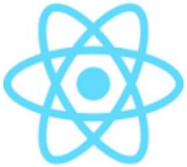
```
class Testing extends React.Component {
  state = { counter: 1 };
  constructor(props) { super(props);
    setTimeout(() => this.tick(), 1000);
    setTimeout(() => this.tick(), 2000); }
  tick() {
    this.setState( (prevState) => ({ counter:
prevState.counter + 1 }) );
  }
  render() { return <h1>Testing,
{this.state.counter}!</h1>; } }
const element = <Testing />;
```

State update using props



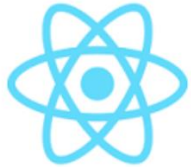
```
class Testing extends React.Component {
  state = { counter: 1, text : '1' };
  constructor(props) {
    super(props);
    this.timer = setInterval(()=> this.tick(), 1000); }
  tick() {
    this.setState((prevState, props) => ({
      counter: ++prevState.counter, text:
prevState.text + props.conj + prevState.counter
    }));
    this.state.counter===3 &&
clearInterval(this.timer); }
  render(){return <h1>Testing,{' '}
{this.state.text}!</h1>;}
}
const element = <Testing conj=" anda " />;
```

State update on condition



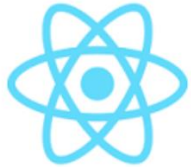
```
class Eat extends React.Component {
  state = { items: [{ product: "apples", q: 1 },{ product: "bananas", q:
10 },{ product: "cherries", q: 5 }] };
  constructor(props) {
    super(props); this.timer = setInterval(() => this.tick(), 1000); }

  tick() {
    this.setState((prevState, props) => ({
      items: prevState.items.map(
      item => (item.product ==='bananas'?{ ...item, q:item.q - 1} :
item)
      )  }));
      let b = this.state.items[1].q;
      console.log("eating a banana...", b, 'left');
      (b <= this.props.until) && clearInterval(this.timer); }

  render() { return <><h1>Inventory</h1>
     <ul>{this.state.items.map(
     item => <li key={item.product}>{item.product} = {item.q}
</li>)}</ul></>;  }}

const element = <Eat until="5" />;
```

State management packages

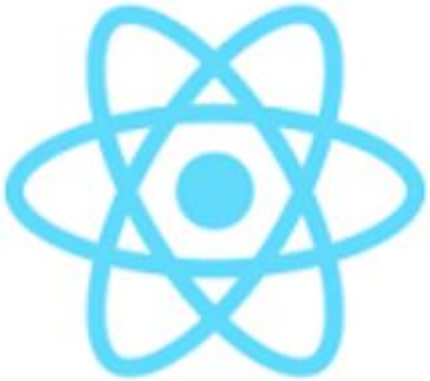


- state management
  - Redux, Hydux, MobX, Apollo Client
  - Easy Peasy – wrapper for Redux
- Now with Hooks and using the Context API, you don't really need to use it as much
- Solves
  - prop drilling
  - predictable state updates through reducers
- Use Hooks aware packages to be easier

Context API

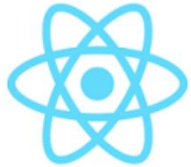


- Provides a way to pass data through the component tree without having to pass props down manually at every level.
- Global state
- Powers Redux
  - Easier to use than Redux



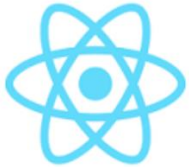
Material-UI

Material-UI



- https://material-ui.com/
  - v4 (Hooks support)
- npm install @material-ui/core
  - @next was for pre-release versions
- Google's 2014 design system implementation in React
- uses react-jss, a CSS-in-JS library
  - by Oleg Isonen
- only loads styles for what it uses
  - Bootstrap loads ALL its CSS

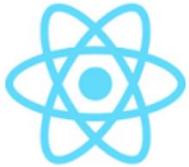
https://unpkg.com/@material-ui/core/umd/material-ui.production.min.js



Adding MUI to CodePen

- JS Gear icon (settings)
  - JavaScript / Add External Scripts/Pens
  - Add package like below
- Browse all packages at https://unpkg.com/@material-ui/core/

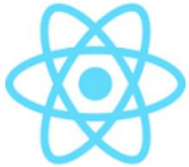
Importing components in projects



- Without { } it has two meanings
  - import Nav.js
  - import Nav/index.js

```
import Nav from './Nav'
```

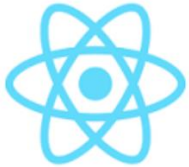
Named imports for project



- Webpack setup allows for tree shaking
  - Tree shaking - import { a class } from @module

```
import { Paper, Typography, TextField } from
'@material-ui/core/';
```

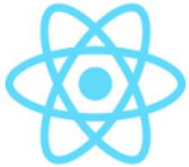
CodePen MUI imports



- Import to CodePen is different than a project
- Import is different from CodePen than previous style as of Feb 2019
  - **old**: const { Button } = window['material-ui'];

```
const { Button } = MaterialUI;
const { MuiThemeProvider, createMuiTheme } =
MaterialUI;
const { colors: {pink, indigo, red} } =
MaterialUI;
```

Roboto font import



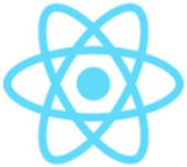
- npm install typeface-roboto –save
  - import 'typeface-roboto'
- CodePen – HTML settings - add to stuff for <head>, or **CSS**

```
<link rel="stylesheet"
href="https://fonts.googleapis.com/css?family=R
oboto:300,400,500">
```

```
@import
url("https://fonts.googleapis.com/css?family=Ro
boto:300,400,400i,700");
```

```
* {font-family: Roboto, sans-serif}
```

https://material.io/tools/icons/?style=baseline



Material icons – import to project

- 900+ icons, best from web font

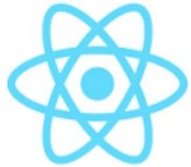
```
<link rel="stylesheet"
href="https://fonts.googleapis.com/icon?family=
Material+Icons">
```

or

```
@import
url("https://fonts.googleapis.com/icon?family=M
aterial+Icons" );
```

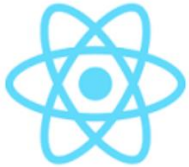
```
<i class="material-icons">face</i>
```

Material icons - classes



```
.material-icons.md-18 { font-size: 18px; }
.material-icons.md-24 { font-size: 24px; }
.material-icons.md-36 { font-size: 36px; }
.material-icons.md-48 { font-size: 48px; }

.material-icons.md-dark { color: rgba(0, 0, 0, 0.54);}
.material-icons.md-dark.md-inactive { color: rgba(0, 0, 0, 0.26); }
.material-icons.md-light { color: rgba(255, 255, 255, 1); }
.material-icons.md-light.md-inactive { color: rgba(255, 255, 255, 0.3); }

const element = <i className="material-icons md-48 md-dark">local_cafe</i>;
```

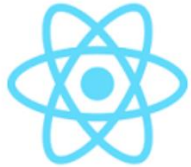
https://material.io/resources/icons



Material Icons

```
const element = (<>
  <i className="material-icons md-18">face</i> <br />
  <i className="material-icons md-24">face</i> <br />
  <i className="material-icons md-36">face</i> <br />
  <i className="material-icons md-48">face</i> <br />
  <i className="material-icons md-48 md-dark">face</i>
<br />
  <i className="material-icons md-48 md-dark md-
inactive">face</i><br />
  <i style={{ backgroundColor: "black" }} className =
"material-icons md-48 md-light">face</i><br />
  <i style={{ backgroundColor: "black" }} className =
"material-icons md-48 md-light md-inactive">face</i>
<br />
</> );
```

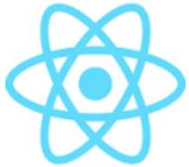
Style attributes



- use object to map CSS rules in JSX eval block
- JavaScript properties don't need to be quoted
- Sometimes, px is not needed but is a good practice
  - margin: '0 5 0 10' doesn't work

```
style={{'margin': '10px'}}
style={{ margin: '10px' }}
style={{ margin: 10 }}
style={{color: 'white', backgroundColor:
'green'}}
```

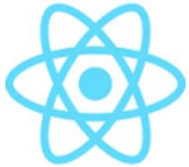
Typography



- https://material-ui.com/api/typography/
- (theme) color - 'default', 'error', 'inherit', 'primary', 'secondary', 'textPrimary', 'textSecondary'

```
const {Typography} = MaterialUI; // and so on…

const element = <Typography variant='h1'
color='textPrimary'>
      Home Page
</Typography>
```

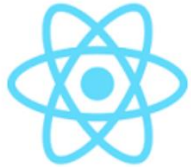
Typography



- variant – style/component for root, default <p>
  - 'h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'subtitle1', 'subtitle2', **'body1**', 'body2', 'caption', 'button', 'overline', 'srOnly', 'inherit'
- align - 'inherit', 'left', 'center', 'right', 'justify'

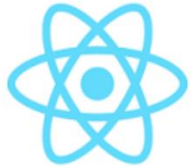
```
const element = <Typography variant='body2'
color='textPrimary' align='center'
gutterBottom>
      Lorem ipsum dolor sit amet
</Typography>
```

Typography



- **booleans** – gutterBottom, noWrap, paragraph
- **component** – any other element that should be what is used for the root
  - for maintaining style to stylesheet rules
  - For switching to inline element.

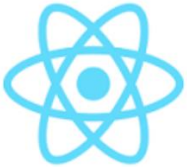
Button



- variant - 'text', 'outlined', **'contained**'
- color – 'default', 'inherit', 'primary', 'secondary'
- booleans - disabled, fullWidth
- size - 'small', 'medium', 'large'
- HTML - href, type

```
const element = <Button variant="contained"
color="primary" fullWidth={true} >Hello
World</Button>;
```

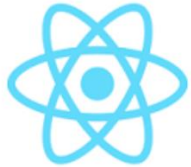
Paper



- component – the element of the root node
  - a base style
- elevation – shadow depth, 0 – 24
- square – rounded corners are default (false)
  - hardly noticeable

```
const element = <Paper component='blockquote'
elevation={6} square={true}>
  <Typography variant='h2' gutterBottom >Paper
Project</Typography>
  <Button variant='outlined' style={{margin:
"-10px 0 5px 8px" }}>Enter</Button>
</Paper>;
```

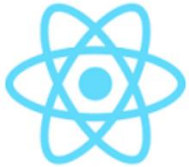
List



- Used for a container of ListItems
- component – base node
- subheader – subheader node
- booleans – dense, disablePadding (vert.)

```
const element =  <List
style={{maxWidth:'20rem',
backgroundColor:'hsl(15, 80%, 80%)'}}>
</List>;
```

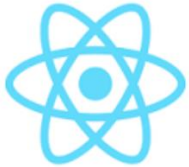
ListItem



- alignItems - 'flex-start', 'center'
- component – base style (li or div based on button)
- booleans – autofocus, dense, button, disabled, disableGutters, divider, selected

```
const element = <><List style={{maxWidth:'20rem',
backgroundColor:'hsl(15, 80%, 80%)'}}>
   <ListItem>Not clickable</ListItem>
   <ListItem button>Clickable button.</ListItem>
</List>
<List component="nav">
   <ListItem button>New list in a nav</ListItem>
</List></>;
```

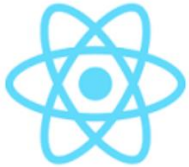
ListItemText



- primary, secondary – text to show
- inset – boolean, ident, use when no icon
- other booleans – disableTypography
- primaryTypographyProps, secondaryTypographyProps – style objects

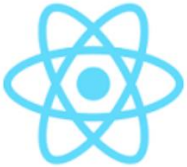
```
const element =   <List style={{maxWidth:'20rem',
backgroundColor:'coral'}}>
  <ListItem button>
    <ListItemText primary = 'Primary text' secondary
= {new Date().toLocaleString()}
secondaryTypographyProps = {{color:'primary'}}
inset/>
  </ListItem> </List>;
```

ListItemIcon



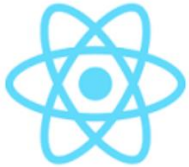
```
const element =   <List style={{ maxWidth: "20rem",
backgroundColor: "coral" }}>
<ListItem button>
  <ListItemIcon>
     <i className="material-icons md-
48">account_circle</i>
  </ListItemIcon>
  <ListItemText primary="Primary person" />
</ListItem>
<ListItem button>
  <ListItemText secondary="Next person" />
  <ListItemIcon>
   <i className="material-icons md-
48">account_box</i>
  </ListItemIcon>
</ListItem></List>;
```

ListItem as a link



```
const element = <>
<Typography variant="h6">Cars</Typography>
<List dense style={{ maxWidth: "20rem",
backgroundColor: "coral" }}>
<ListItem style={{ maxHeight: "3rem"}} button
component="a"
href='https://www.audiusa.com/models/audi-e-tron'>
   <ListItemIcon style={{color:'white'}}><i
className="material-icons md-48">directions_car</i>
</ListItemIcon>
    <ListItemText primary="Audi e-tron" />
</ListItem>
<ListItem style={{ maxHeight: "3rem"}} button
component="a" href='https://www.hyundaiusa.com/kona-
electric/index.aspx'>
   <ListItemIcon><i className="material-icons md-
48">directions_car</i></ListItemIcon>
   <ListItemText primary="Hyundai Kona" />
</ListItem> </List> </>;
```

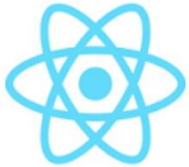
Avatar



- alt, component, imgProps, sizes, src, srcSet

```
const element = <> <Typography variant="h6">Cars</Typography>
<List dense style={{ maxWidth: "20rem"}}>
<ListItem button>
<Avatar style={{marginRight:'.5rem'}}><i className="material-
icons md-12">directions_car</i></Avatar>
<ListItemText primary="Audi e-tron" /></ListItem>
<ListItem button>
<Avatar style={{fontSize:'.8rem', marginRight:'.5rem',
color:'red'}}>HK</Avatar>
<ListItemText primary="Hyundai Kona" /></ListItem>
<ListItem button>
<Avatar
src='https://d3g9pb5nvr3u7.cloudfront.net/sites/539a28913f3c0
fd71ed4e43c/-1406957656/256.png'
style={{marginRight:'.5rem'}}></Avatar>
<ListItemText primary="Tesla Model 3" />
</ListItem></List></>;
```

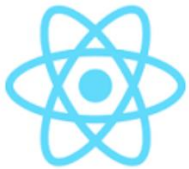
Divider



- variant – fullWidth, inset, middle
- component – the base element
- booleans – light, absolute

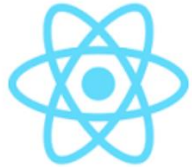
```
const element = <> <Typography variant="h4">Cars</Typography>
    <Divider />
    <List style={{ maxWidth: "20rem" }}>
    <ListItem button ><ListItemText inset primary="Audi e-
tron" /></ListItem>
    <Divider variant='inset' />
    <ListItem button><ListItemText primary="Hyundai Kona"
/></ListItem>
    <Divider variant='middle' />
    <ListItem button><ListItemText primary="Tesla Model 3"
/></ListItem>
    <Divider /> </List></>;
```

Card, CardMedia, CardContent, CardActions



```
const element= <><Card style={{ maxWidth:"20rem" }}>
   <CardMedia style={{ height: 0, padding: '40%
80%'}}  image="https://airplantstore.com/wp-
content/uploads/2018/10/IMG_2968-e153974470433.jpg"
title="shells"/>
   <CardContent>
   <Typography gutterBottom variant="headline"
component="h2">Unicorn Shells</Typography>
   <Typography component="p">Marine snails having a
prominent spine on the lip of the shell</Typography>
   </CardContent>
   <CardActions><Button size="small"
color="primary">Go to Topic</Button></CardActions>
</Card></>;
```

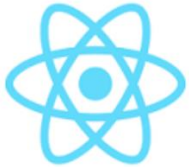
CardHeader, IconButton



- nodes - action, avatar, subheader, title
- props –subheaderTypographyProps, titleTypographyProps
- booleans - disableTypography


```
<Card style={{ maxWidth: "20rem" }}>
      <CardHeader
        avatar={<Avatar>SL</Avatar>}
        action={<IconButton><i className="material-
icons md-12">beach_access</i></IconButton>}
        title="Sea Life"
        subheader="September 14, 2016"/>
<CardMedia…
```

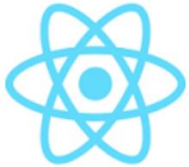
https://material.io/resources/icons



Icon, SvgIcon, Font Awesome

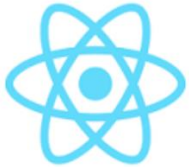
```
const element= <>
<Icon color={'primary'}
style={{fontSize:'128px'}}>star</Icon>
<Icon color={'secondary'} className='material-
icons md-48'>arrow_back</Icon>
<SvgIcon>
    <path d="M20 12l-1.41-1.41L13 16.17V4h-
2v12.17l-5.58-5.59L4 12l8 8 8-8z" />
</SvgIcon>
<i className='far fa-hand-point-left fa-5x' />
</>;
```

Page layout templates



- Templates to use
  - https://material-ui.com/getting-started/page-layout-examples/
  - https://themes.material-ui.com/
  - https://themeforest.net/tags/material%20ui

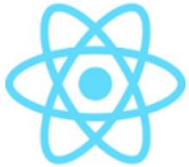
Grid



- based on Flexbox
- two types – container, item
- set widths in %
- RWD breakpoints – xs, sm, md, lg, xl

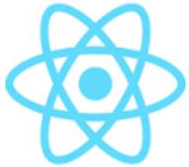
```
 <Grid container style={{border: '2px solid red', padding: 5}} >
   <Grid item style={{border: '1px solid gray'}}>small component</Grid>
   <Grid item style={{border: '1px solid gray', padding: 20}}>another small component   </Grid>
</Grid>
```

Data for examples



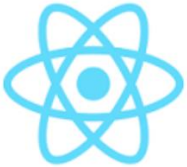
```
const imageURL =
'https://images.unsplash.com/photo-1561266436-
05386f8c5a98?ixlib=rb-1.2.1';
const data = [
{img: imageURL, title: 'Image 1',  author:
'author 1', cols: '1'},
{img: imageURL, title: 'Image 2',  author:
'author 2', cols: '2'},
{img: imageURL, title: 'Image 3',  author:
'author 3', cols: '2'},
{img: imageURL, title: 'Image 4',  author:
'author 4', cols: '1' }
];
```

GridList, GridListTile, GridListTileBar, ListSubheader



```
const element= <GridList cellHeight={250} cols={3}
style={{maxWidth:'900px'}}>
    <GridListTile key="Subheader" cols={3} style={{
height: 'auto' }}>
        <ListSubheader component="div"><Typography
variant='h3'>Photos</Typography></ListSubheader>
    </GridListTile>
  {data.map(tile => (
  <GridListTile key={tile.author} cols={tile.cols ||
1}>
   <img src={tile.img} alt={tile.title} />
   <GridListTileBar
      title={tile.title}  subtitle={<span>by:
{tile.author}</span>}
      actionIcon={ <IconButton title={`info about
${tile.title}`}><Avatar>Img</Avatar></IconButton>} />
   </GridListTile>))}
</GridList>;
```

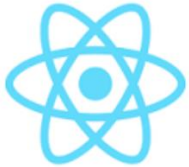
Table, TableBody, TableCell, TableHead, TableRow,



- Complex but feature-rich
  - https://material-ui.com/components/tables/

```
const element =  <Paper><Table>
<TableHead><TableRow>
  <TableCell component=''>Title</TableCell>
<TableCell>Author</TableCell> <TableCell
align="right">Columns</TableCell>
</TableRow></TableHead>
<TableBody>{data.map(row => (
   <TableRow key={row.author}>
      <TableCell>{row.title}</TableCell>
      <TableCell>{row.author}</TableCell>
      <TableCell align="right">{row.cols}</TableCell>
   </TableRow>))}
</TableBody>
</Table></Paper>;
```

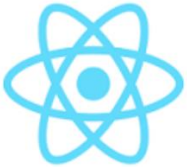
TextField



- defaultValue, helperText, label, name, value
- variant – standard, outlines, filled
- margin – dense, none, normal
- booleans – autofocus, disabled, error, fullWidth, multiline (rows, rowsMax), required, select

```
const element= <> <TextField label="Brand"
placeholder="Brand" name="brand"/><br/>
<TextField label="Notes" placeholder="Notes"
name="model" multiline />
</>;
```

https://material-ui.com/components/text-fields/

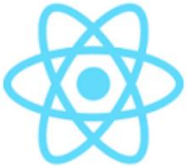


TextField

- For html attributes not implemented use inputProps

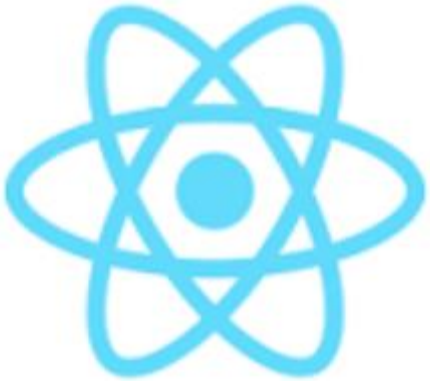
```
<TextField label="Notes" placeholder="Notes"
name="model" helperText='write anything you
like' multiline
  inputProps={{maxlength:'5'}}
/>
```

https://material-ui.com/components/pickers/



Date and time pickers

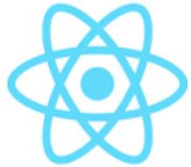
```
const today = new Date();
const tomorrow = new
Date(today.getTime()+1000*60*60*24);

const element= <> <Paper style={{padding:10,
margin:10}}>
<form noValidate>
<TextField id="date" label="Due date" type="date"
defaultValue={tomorrow.toISOString().slice(0,10)}
InputLabelProps={{shrink: true, }}/>
</form>
</Paper> </>;
```



Lifecycle management

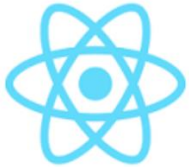
Events

Task splitting



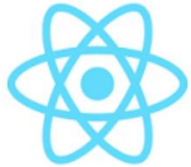
- State management
  - Events update state
  - Determine changed data
- State rendering
  - State updates UI (JSX)
  - Adapt JSX to new data

Events



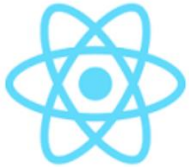
- HTML used all lowercase
  - onclick='doSomething()'
- JSX uses camelCase
  - onClick = {doSomething}
- no returning false;
  - use preventDefault( )

Event handling



- Write a component event handler
- Assign it to an event attribute of the element it listens to.
  - onClick={this.handleClick}
- Bind. Class methods are not bound by default.
  - Solutions
    - Use bind in
      - constructor
      - non-class function
    - *Use experimental syntax with lambda

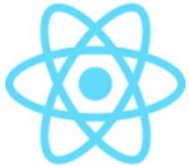
Binding in constructor



- Allows access to state and other vars

```
constructor(props) {
      super(props);
      this.onClick = this.onClick.bind(this);
}
```

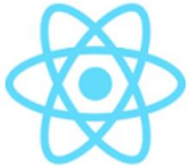
Binding by lambda



- Class props syntax in Create React App

```
const handleClick = () =>
{console.log('clicked');}

const element = <Button variant='contained'
color='secondary' onClick={handleClick}
>Button</Button>;
```

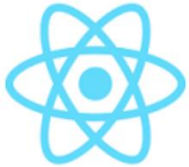
Inline event handler



- Instead of binding to a named function, you can inline the function definition with a lambda

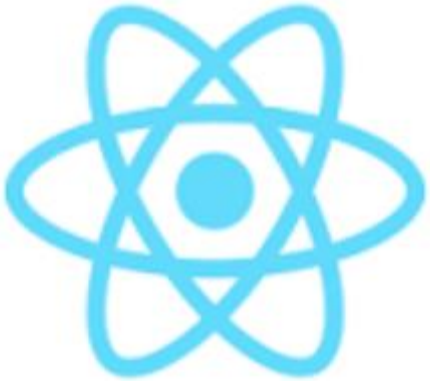
```
const element = <Button
      onClick={e => {
        console.info("Synthetic event:", e);
      }}
      variant="contained"
      color="primary"
    >Log something</Button>;
```

Sending arguments to event handlers



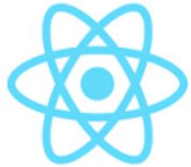
- Change the onClick binding to a lambda function
- Add as many parameters as you need to capture the info. We create a new function here.

```
const element = <Button variant="contained"
color="primary" onClick={(e) =>
this.handleClick(e, 'abc', 123)}>Log 3
things</Button>;


handleClick = (e, arg1, arg2) =>
{console.log(e, arg1, arg2);};
```



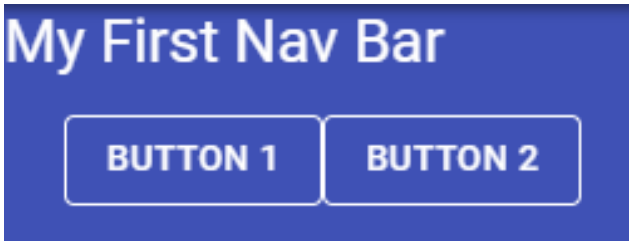
Event-driven MUI

AppBar, Toolbar

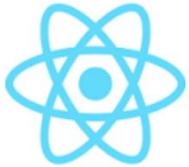


- AppBar stacks content
- ToolBar inlines them

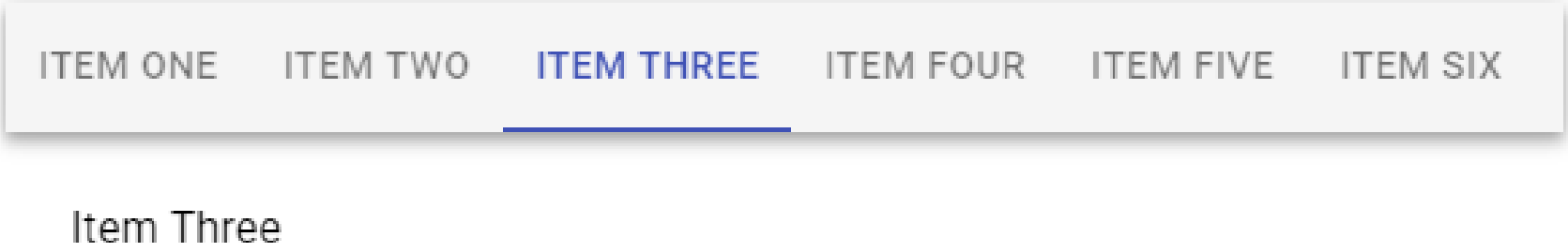
```
const element = <AppBar position="static">
  <Typography variant="h5" >
    My First Nav Bar
  </Typography>
  <Toolbar>
    <Button variant="outlined"
color="inherit">Button 1</Button>
    <Button variant="outlined"
color="inherit">Button 2</Button>
  </Toolbar>
</AppBar>;
```



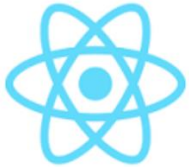
Tabs



- Example: https://codesandbox.io/s/qlq1j47l2w
- But, it's better to use the router to allow paths to be put on the history.



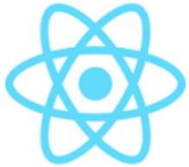
Snackbar



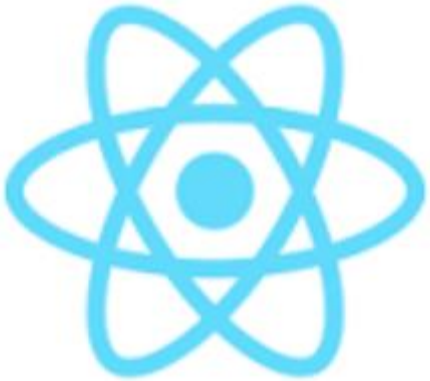
- Requires
  - state fields of **open**, **message**
  - two events of a trigger and **handleClose**

```
class Sb extends React.Component {
  state={open: true, message:'a message for 5
secs'};
  handleClose = (e) => {
    this.setState({ open: false });
  };
  render() {return <Snackbar style = {{width: 300,
color: 'green'}}  open={this.state.open}
onClose={this.handleClose} autoHideDuration={5000}
message={this.state.message} />; }
}
const element =  <Sb/> ;
```

Dialog

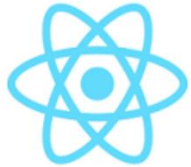


```
function DialogTest() {
    const [open, setOpen] = React.useState(false);
    const handleClose = () => {setOpen(false);};
    const handleClickOpen = () => {setOpen(true); };
    return (<>
     <Button variant="outlined" color="primary" onClick={handleClickOpen}>Open alert
dialog</Button>
    <Dialog open={open} onClose={handleClose}>
    <DialogTitle id="alert-dialog-title">
       {"Use Google's location service?"}
    </DialogTitle>
    <DialogContent>
      <DialogContentText id="alert-dialog-description">
Let Google help apps determine location. This means sending anonymous location data to
Google, even when no apps are running.
      </DialogContentText>
    </DialogContent>
    <DialogActions>
    <Button onClick={handleClose} color="primary"> Disagree</Button>
    <Button onClick={handleClose} color="primary" autoFocus>
Agree</Button>
    </DialogActions> </Dialog> </>);}

const element = <DialogTest />;
```



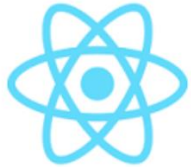
Forms

State is required



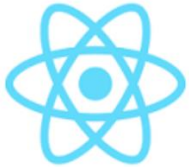
- Some form elements do not update without state
  - textarea
  - checkbox
  - radio buttons

Event handlers



- onChange
  - for responsive textarea, checkbox and radio components
- onBlur
  - for text fields

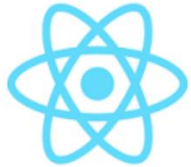
defaultValue, defaultChecked



- The text field attribute **value** usually holds a default value.
  - overridden by the read-only jsx value attribute
- Use defaultValue='some default value' instead.
- The textarea can use the attribute value.
- defaultChecked – radio buttons and checkboxes

```
<input type="checkbox" defaultChecked ='?'>
<input type="radio" defaultChecked ='?'>
<select defaultValue ='?'>
<textarea defaultValue ='?'>
```

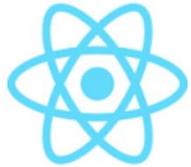
Controlled components



- Components that get and set their value through the state object.
  - A single source of truth
  - The authority

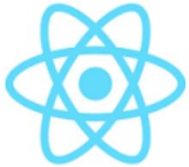
```
<input type="text" value={this.state.value}
onChange={this.handleChange} />
```

Controlled components



- Managing the form data through the state object
  - value property is where the element saves data
  - you usually ask the element for its value
- Instead, use the this.state object
  - or hook controlled variable
- Controlled are recommended.

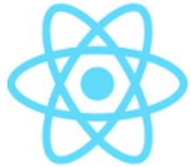
Select values



- Use a value attribute only for the select parent
- The children options will be matched and selected.

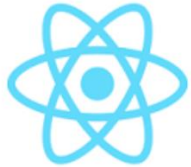
```
<select value={this.state.value}
onChange={this.handleChange}>
      <option value="grape">Grape</option>
      <option value="lime">Lime</option>
```

Object schema validation & parsing



- Yup – front end browser based
- Joi – server side
  - Object schema description language and validator for JavaScript objects
  - https://github.com/hapijs/joi
- data security
- readability

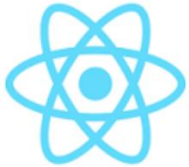
Form values



- Set initial value in constructor
- Update value in event handler
- Show value from state in value property

```
// constructor
this.state = {value: ''};
// event handler
this.setState({value: event.target.value});
// jsx
<input type="text" value={this.state.value}
onChange={this.handleChange} />
```

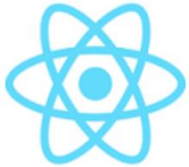
textarea



- No difference from input/text
- HTML puts value as body

```
<textarea value={this.state.value}
onChange={this.handleChange} />
```

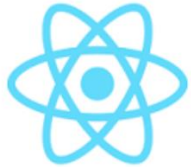
select



- HTML uses a selected attribute on the option
- Grouped options in a select parent
- React uses the value attribute again.

```
<select value = {this.state.value}
onChange = {this.handleChange}>
      <option…
      <option…
      <option…
      <option…
```

Name the fields



- Using names for fields allows for detection in the event handler
  - Material-UI uses the ID if you don't have a name
- Checkboxes require a different value

```
// jsx
<input name = 'thisData'

// handler
const name = event.target.name;
const value = target.type === 'checkbox' ?
target.checked : target.value;
this.setState({ [name]: value });
```

Questions and Answers