NODE PROGRAM

LESSON 1: REACT BASICS

REACT.JS VERSION: 0.14.3
LAST UPDATED: FEB 2016

QUICK INTRO

THE DEFINITION

WHAT IS REACT.JS?

IT'S NOT ABOUT TEMPLATES, OR DATA BINDING, OR DOM MANIPULATION. IT'S ABOUT USING FUNCTIONAL PROGRAMMING WITH A VIRTUAL DOM REPRESENTATION TO BUILD AMBITIOUS, HIGH-PERFORMANCE APPS WITH JAVASCRIPT.

PETE HUNT HTTP://BIT.LY/1U53RB2

KEY DIFFERENCES

- VIRTUAL DOM
- DECLARATIVE (NOT IMPERATIVE)
- FUNCTIONAL
- NO DOM MANIPULATION
- NO TEMPLATES
- NO EVENT LISTENERS OR HANDLERS

MVC

REACT.JS IS NOT A MODEL-VIEW-CONTROLLER (MVC) FRAMEWORK/LIBRARY.

YOU NEED TO BRING YOUR OWN MODELS AND ROUTERS.

REACT.JS IS ONLY VIEW.

WEB STACK

REACT.JS CAN WORK WITH OTHER MVC-LIKE FRAMEWORK SUCH AS BACKBONE.JS AND

REACT.JS IS OFTEN USED WITH FLUX AND METEOR.

HELLO WORLD

DOWNLOADING REACT.JS

THERE ARE SEVERAL WAYS:

1. SOURCE CODE FOR THIS COURSE.
2. REACT.JS WEBSITE: HTTP://FACEBOOK.GITHUB.IO/REACT/DOWNLOADS.HTML.
3. NMP: `$ npm install react react-dom`.

NPM

```
$ mkdir react-project
$ cd react-project
$ npm init

$ npm i --save react@0.14.7 react-dom@0.14.7
```

react-project/node_modules/react/dist/
react.js
react-project/node_modules/react-dom/dist/
react-dom.js

CREATING HTML

START THE HTML FILE `hello-world.html`:

```
<!DOCTYPE html>
<html>
```

REACT.JS INCLUSION

INCLUDE REACT.JS AND REACT DOM LIBRARIES:

```
<head>
  <script src="react.js"></script>
  <script src="react-dom.js"></script>
</head>
```

IF YOU DOWNLOADED THE REACT-0.14.3, THEN THIS FILE WILL BE IN THE BUILD FOLDER.
YOU CAN CHOOSE MINIFIED VERSION (REACT.MIN.JS) AS WELL.

```
<head>
  <script src="node_modules/react/dist/react.js"></script>
  <script src="node_modules/react-dom/dist/react-dom.js"></script>
</head>
```

REACT.JS CDN

OR HOTLINK TO FACEBOOK CDN:

```
<script src="https://fb.me/react-0.14.7.min.js"></script>
<script src="https://fb.me/react-dom-0.14.7.min.js"></script>
```

HTML STRUCTURE

THE `div` IN THE BODY OF `hello-world.html`:

```
  <body>
    <div id="example"></div>
    <script type="text/javascript">
    ... // React.js code
    </script>
  </body>
</html>
```

H1 ELEMENT

THE FOLLOWING SNIPPET CREATES THE H1 REACT.JS OBJECT WITH CONTENT 'HELLO WORLD!':

```
React.createElement('h1', null, 'Hello world!')
```

RENDERING

ONCE THE ELEMENT IS CREATED WE RENDER IT TO THE DOM ELEMENT WITH ID EXAMPLE AND RENDERS IT:

```
ReactDOM.render(React.createElement('h1', null, 'Hello world!'),
  document.getElementById('example')
);
```

RUNNING THE PAGE

OPEN `hello-world.html` AND CHECK IF YOU SEE HELLO WORLD!

INSPECTING HTML

IF WE INSPECT THE HTML GENERATED BY REACT.JS, IT WILL HAVE THIS ATTRIBUTE:

```
<h1 data-reactid=".0">Hello world!</h1>
```

HTML ARGUMENTS

VIRTUALLY ALL HTML ARGUMENTS ARE SUPPORTED SO YOU CAN PASS THEM LIKE THIS:

```
React.createElement("div", {style: "color:red"}, "Hello ", "world!")
```

NOTE: YOU CAN ADD MANY PARAMETERS AT THE END TO COMBINE THEM.

FOR AND CLASS ATTRIBUTES

IF YOU NEED TO USE `for` OR `class` ATTRIBUTES, THEIR NAMES ARE `forHtml` AND `className`. FOR EXAMPLE.

```
React.createElement("div", {className: "hide"}, "Hello world!")
```

MEET JSX

WHAT IS JSX

JSX IS A COMBINATION OF JAVASCRIPT AND XML. HTML IS A FORM OF XML:

```
ReactDOM.render(
  <h1>Hello world!</h1>,
  document.getElementById('example')
);
```

WHAT IS JSX (CONT.)

JSX IS COMPILED INTO NATIVE/REGULAR JAVASCRIPT. JSX ALLOWS FOR EASIER AND FASTER WRITING HTML VIEWS AND ELEMENTS ALONG WITH JAVASCRIPT

HTTPS://JSX.GITHUB.IO/

WHY USE JSX

THE MIX OF XML AND JS LOOKS WEIRD. BUT ITS THE RECOMMENDED WAY OF WRITING REACT.JS APPS BECAUSE IT PROVIDES SYNTAX FOR COMPONENTS, LAYOUTS AND HIERARCHY.

NOTE: AS YOU'VE SEEN FROM THE PREVIOUS HELLO WORLD! EXAMPLE, JSX IS OPTIONAL.

WAYS TO USE JSX

1. PRE-PROCESS WITH `babel-cli`: PRODUCTION RECOMMENDED
2. BUILD WITH GULP, GRUNT, WEBPACK AND BABEL: PRODUCTION RECOMMENDED
3. RUN-TIME VIA `babel-standlone` V6.4.4: DEVELOPMENT ONLY

RUN-TIME JSX

RUN-TIME JSX PERFORMED BY `babel-standlone` V6.4.4 FILE (BABEL V5.X HAD `browser.js`).

DOWNLOAD HTTPS://CDNJS.CLOUDFLARE.COM/AJAX/LIBS/BABEL-STANDALONE/6.4.4/BABEL.MIN.JS

BABEL STANDALONE

FOR NOW, WE'LL USE THE BABEL STANDALONE V6.4.4. THIS CODE GOES INTO `hello-world-jsx.html`:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-standalone/6.4.4/babel.min.js"></script>
```

JSX TYPE SCRIPT

LET'S CONVERT THE CODE FROM JAVASCRIPT TO JSX BY CHANGING `text/javascript` TO `text/babel`:

```
  <body>
    <div id="example"></div>
    <script type="text/babel">
    ...
    </script>
  </body>
</html>
```

JSX CODE

CHANGE `React.createElement` TO `<h1>...</h1>`:

```
ReactDOM.render(
  <h1>Hello world!</h1>,
  document.getElementById('example')
);
```

RUNNING THE CODE

OPEN `hello-world-jsx.html` AND CHECK IF YOU SEE HELLO WORLD!

BABEL

THE JAVASCRIPT COMPILER.

HTTP://BABELJS.IO/

COMPILING JSX WITH BABEL

FOR MORE REALISTIC AND PRODUCTION-LIKE EXAMPLE. WE'LL USE BABEL. THIS WILL ALLOW US TO COMPILE JSX INTO NATIVE JS AND RUN ONLY NATIVE JS IN THE BROWSER. THIS WILL INCREASE PERFORMANCE IN PRODUCTION REACT.JS APPS.

WHAT IS BABEL

BABEL ALLOWS YOU TO USE ECMASCRIPT 6 NOW BY COMPILING ES6 CODE INTO ES5-FRIENDLY CODE TO SUPPORT BROWSERS THAT DON'T HAVE ES6 YET.

HTTPS://BABELJS.IO/

SEPARATION OF CONCERNS

FIRSTLY, LET'S ABSTRACT JSX CODE FROM `hello-world-jsx.html` HTML INTO TWO FILES:

1. `hello-world.jsx`
2. `hello-world-jsx-babel.html`

NOTE: THE NATIVE JAVASCRIPT EXAMPLE CAN ALSO BE SPLIT INTO TWO FILES.

CODE INCLUSION

ADD THIS LINE TO THE `hello-world-jsx-babel.html` FILE RIGHT BEFORE CLOSING `body`:

```
...
    <script src="hello-world.js"></script>
...
```

PRE-PROCESSED JSX

PRE-PROCESSING JSX IS BETTER OVER RUN-TIME BECAUSE IT'S FASTER. PRE-PROCESSING IS THE SAME AS COMPILING INTO NATIVE JAVASCRIPT. YOU CAN DO IT WITH BABEL CLI

BABEL CLI

```
$ npm install --save-dev babel-cli@6.4.4 babel-preset-react@6.4.4
$ echo '{ "presets": ["react"] }' > .babelrc
$ ./node_modules/.bin/babel src -d lib
```

```
$ ./node_modules/.bin/babel hello-world.jsx -o hello-world.js -w
```

THE `hello-world.js` WILL BE CREATED. LEAVE BABEL RUNNING SO THE `-w` CAN UPDATE CHANGES AUTOMATICALLY.

COMPILING WITH BABEL

COMPILE JSX INTO REGULAR JS WITH:

```
$ babel -w hello-world-component.jsx -o
hello-world-component.js
```

RUNNING THE CODE

OPEN `hello-world-jsx-babel.html` AND CHECK IF YOU SEE HELLO WORLD!

AUTO UPDATE

EACH TIME YOU CHANGE `hello-world.jsx`, THE TOOL SHOULD UPDATE `hello-world.js` WITH THE MESSAGE:

```
change hello-world.jsx
```

COMPOSABLE COMPONENTS

THE CONCEPT OF COMPONENTS IS THE FOUNDATION OF REACT.JS PHILOSOPHY. THEY ALLOW YOU TO REUSE CODE AND LOGIC. THEY ARE LIKE TEMPLATES ONLY BETTER.

TYPES OF REACT.JS COMPONENTS

REACT.JS COMPONENT TYPES:

- REGULAR HTML ELEMENTS SUCH AS H1, P, DIV, ETC.
- CUSTOM OR COMPOSABLE COMPONENTS

DIFFERENCE BETWEEN REGULAR AND CUSTOM COMPONENTS

IF IT'S A REGULAR HTML TAG NAME, THEN REACT.JS WILL CREATE SUCH ELEMENT. OTHERWISE, IT WILL LOOK FOR THE CUSTOM COMPONENT DEFINITION.

NOTE: REACT.JS USES LOWER-CASE VS. UPPER CASE TO DISTINGUISH BETWEEN HTML TAGS AND COMPONENTS.

DEFINING A COMPONENT

COMPOSABLE COMPONENTS ARE CREATED WITH `React.createClass` AND MUST HAVE `render` METHOD THAT RETURNS REGULAR COMPONENT (DIV, H1, ETC.):

```
var HelloWorld = React.createClass({
  render: function() {
    return <h1>Hello world!</h1>
  }
})
```

REFACTORING WITH A HELLOWORLD COMPONENT

THE `hello-world-component.jsx` FILE HAS A CUSTOM COMPONENT:

```
var HelloWorld = React.createClass({
  render: function() {
    return (
      <h1>Hello world!!!</h1>
    )
  }
})

ReactDOM.render(
  <HelloWorld/>,
  document.getElementById('example')
)
```

HTML SKELETON

POINT YOUR `hello-world-component.html` **TO USE** `hello-world-component.js`. **NOT** `hello-world-component.jsx`:

```
<!DOCTYPE html>
<html>
  <head>
    <script src="react.js"></script>
  </head>
  <body>
    <div id="example"></div>
    <script src="hello-world-component.js"></script>
  </body>
</html>
```

RUNNING THE CODE

OPEN `hello-world-component.html` AND CHECK IF YOU SEE HELLO WORLD!

AUTO UPDATE

EACH TIME YOU CHANGE `hello-world-component.jsx`. THE TOOL SHOULD UPDATE `hello-world.js` WITH THE MESSAGE:

```
change hello-world.jsx
```

NESTED ELEMENTS

NESTING REACT.JS COMPONENTS IS EASY.

RENDERING TITLE AND TEXT

THIS IS HOW WE CAN NEST `h1` AND `p` INSIDE OF `div`:

```
ReactDOM.render(
  <div>
    <h1>
      Core React.js
    </h1>
    <p>This text is very useful for learning React.js.</p>
  </div>,
  document.getElementById('example')
)
```

SINGLE TOP-LEVEL TAG

REMEMBER TO ALWAYS HAVE ONLY ONE ELEMENT AS THE TOP LEVEL TAG!

FOR EXAMPLE, THIS IS A NO GO:

```
ReactDOM.render(
  <h1>
    Core React.js
  </h1>
  <p>This text is very useful for learning React.js.</p>
  ,
  document.getElementById('content')
)
```

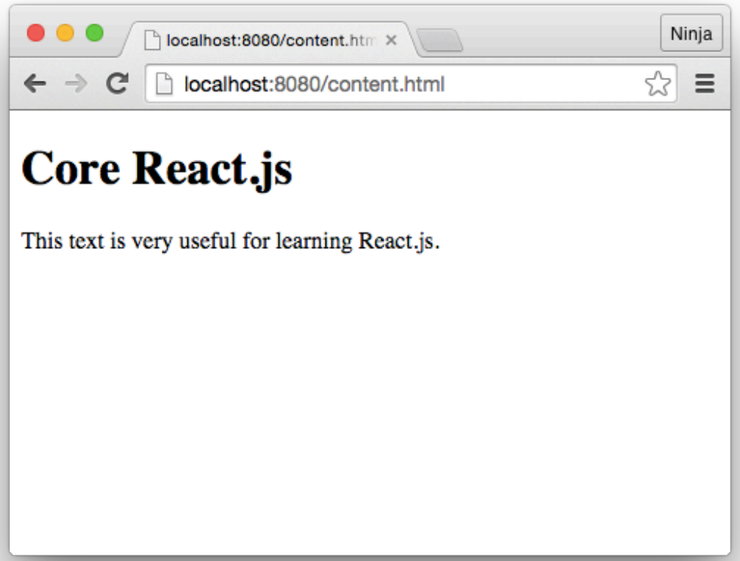
OBVIOUSLY, WE CAN CREATE NESTED STRUCTURES IN CUSTOM COMPONENTS:

```
var Content = React.createClass({
  render: function() {
    return (
      <div>
        <h1>
          Core React.js
        </h1>
        <p>This text is very useful for learning React.js.</p>
      </div>
    )
  }
})
```

```
ReactDOM.render(
  <Content />,
  document.getElementById('content')
)
```

RUNNING THE CODE

OPEN `content.html` AND CHECK IF YOU SEE TITLE AND TEXT!



ORDER OF THE CODE

REMEMBER THAT THE CONTENT ELEMENT (`<div id="content"></div>`) MUST PRECEDE THE REACT.JS CODE (`<script ...`). FOR THE `getElementById` METHOD TO LOCATE THE PROPER DOM ELEMENT:

```
...
    <div id="content"></div>
    <script type="text/jsx">
      var Content = React.createClass({
      ...
      })
      ReactDOM.render(
        <Content />,
        document.getElementById('content')
      )
    </script>
...
```

VARIABLES

USE {} TO RENDER VARIABLE INSIDE OF JSX:

```
{a}
{' '}
{b}
```

VARIABLE EXAMPLE

IN THE `variable/script.jsx` FILE. WE OUTPUT THE VALUE OF `a`:

```
var Content = React.createClass({
  render: function() {
    var a = 1
    return (
      <div>
        <h1>
          {a}. Core React.js
        </h1>
        <p>This text is very useful for learning React.js.</p>
      </div>
    )
  }
})
```

VARIABLE DEMO

PLNKR ALLOWS TO EDIT, RUN AND PREVIEW CODE IN THE BROWSER.

VARIABLE EXAMPLE: HTTP://PLNKR.CO/EDIT/AKNL72P6AXP71CYLEINJ?P=PREVIEW

STATES

STATES ARE MUTABLE PROPERTIES OF COMPONENTS MEANING THEY CAN CHANGE. WHEN STATE CHANGES THE CORRESPONDING VIEW CHANGES, BUT EVERYTHING ELSE IN DOM REMAINS INTACT.

INITIAL STATE

THE INITIAL STATE IS SET BY THE `getInitialState` METHOD WHICH IS CALLED ONCE WHEN THE ELEMENT IS CREATED.

LET'S USE THIS METHOD TO RETURN `a`:

```
var Content = React.createClass({
  getInitialState: function(){
    return {a: 0}
  },
  ...
```

UPDATING STATE

STATE IS UPDATED WITH `this.setState`. SO THIS CODE WILL UPDATE THE VALUE WITH A RANDOM NUMBER EVERY 300 MILLISECONDS:

```
var Content = React.createClass({
  getInitialState: function(){
    var _this = this
    setInterval(function(){
      _this.setState({a: Math.random()})
    }, 300)
    return {a: 0}
  },
  ...
```

OUTPUTTING THE STATE

TO OUTPUT THE STATE PROPERTY a. WE USE {this.state.a}:

```
  render: function() {
    return (
      <div>
        <h1>Changing the State</h1>
        <p>This value is random: {this.state.a}</p>
      </div>
    )
  }
})
```

RENDERING

THE RENDERING DIDN'T CHANGE:

```
ReactDOM.render(
  <Content />,
  document.getElementById('content')
);
```

HTTP://PLNKR.CO/EDIT/ZWPBX5ORDDV01BO4EHGW?P=PREVIEW

COMPONENT METHODS

CALLING METHODS

IT'S POSSIBLE TO INVOKE COMPONENTS METHODS FROM THE {} INTERPOLATION:

```
var Content = React.createClass({
  getA: function(){
    return 10
  },
  render: function() {
    return (
      <div>
        <p>This value is return by the method: {this.getA()} </p>
      </div>
    )
  }
})
...
```

HTTP://PLNKR.CO/EDIT/UMGFDUHCQFUGLJ8MVSBM?P=PREVIEW

COMPONENT EVENTS

EVENTS

COMPONENTS HAVE NORMALIZED (CROSS-BROWSER) EVENTS SUCH AS

```
onClick onContextMenu onDoubleClick onDrag onDragEnd onDragEnter onDragExit
onDragLeave onDragOver onDragStart onDrop onMouseDown onMouseEnter onMouseLeave
onMouseMove onMouseOut onMouseOver onMouseUp
```

DECLARING EVENTS

REACT.JS IS DECLARATIVE. NOT IMPERATIVE. SO WE WON'T ATTACH EVENT LIKE WE WOULD DO WITH JQUERY. INSTEAD WE DECLARE THEM IN THE JSX AND CLASSES:

```
var Content = React.createClass({
  getInitialState: function(){
    return {counter: 0}
  },
  click: function(e){
    this.setState({counter: ++this.state.counter})
  },
  ...
```

BUTTON ONCLICK EVENT

THE BUTTON HAS THE `onClick={this.click}`.

THE NAME MUST MATCH THE METHOD OF THE Content COMPONENT CLASS:

```
...
  render: function() {
    return (
      <div>
        <button onClick={this.click}>Don't click me {this.state.counter} times!</button>
      </div>
    )
  }
})
```

DEMO

HTTP://PLNKR.CO/EDIT/SIFUS7NG6GKT45T4FVFR?P=PREVIEW

PROPS

PROPS OR PROPERTIES ARE IMMUTABLE MEANING THEY DON'T CHANGE. THEY ARE PASSED BY PARENT COMPONENTS TO THEIR CHILDREN.

USING PROPS

```
var ClickCounterButton = React.createClass({
  render: function() {
    return <button onClick={this.props.handler}>Don't click me {this.props.counter} times! </button>
  }
})
```

SUPPLYING PROPS

PROVIDE PROPS TO THE CLICKCOUNTERBUTTON COMPONENT:

```
var Content = React.createClass({
  getInitialState: function(){
    return {counter: 0}
  },
  click: function(e){
    this.setState({counter: ++this.state.counter})
  },
  render: function() {
    return (
      <div>
        <ClickCounterButton counter={this.state.counter} handler={this.click}/>
      </div>
    )
  }
})
...
```

HTTP://PLNKR.CO/EDIT/0AKIKHI2NC9BTWOTWXHW?P=PREVIEW

WHERE TO PUT LOGIC

IN THIS EXAMPLE, CLICK EVENT HANDLER WAS IN THE PARENT ELEMENT. YOU CAN PUT THE EVENT HANDLER ON THE CHILD ITSELF, BUT USING PARENT ALLOWS YOU TO EXCHANGE INFO BETWEEN CHILDREN COMPONENTS.

LET'S HAVE A BUTTON:

```
var ClickCounterButton = React.createClass({
  render: function() {
    return <button onClick={this.props.handler}>Don't click me! </button>
  }
})
```

EXCHANGING PROPS BETWEEN CHILDREN

THIS IS A NEW COMPONENT WHICH DISPLAYS VALUE PROP:

```
var Counter = React.createClass({
  render: function(){
    return <span>Clicked {this.props.value} times.</span>
  }
})
```

PARENT COMPONENT

THE PARENT COMPONENT PROVIDES PROPS ONE OF WHICH IS A HANDLER:

```
var Content = React.createClass({
  getInitialState: function(){
    return {counter: 0}
  },
  click: function(e){
    this.setState({counter: ++this.state.counter})
  },
  render: function() {
    return (
      <div>
        <ClickCounterButton handler={this.click}/>
        <br/>
        <Counter value={this.state.counter}/>
      </div>
    )
  }
})
```

HTTP://PLNKR.CO/EDIT/ACCOPASRD4ABKS2V1BLX?P=PREVIEW

COMPONENTDIDMOUNT

THE `componentDidMount` METHOD IS INVOKED WHEN COMPONENT IS INSERTED INTO THE DOM. YOU CAN USE THIS METHOD TO PERFORM OPERATIONS, AND/OR SEND AJAX/XHR REQUESTS.

COMPONENTDIDMOUNT EXAMPLE

PRINT DOM:

```
var Content = React.createClass({
  componentDidMount: function(e){
    console.log(ReactDOM.findDOMNode(this))
  },
  render: function() {
    return (
      <div/>
    )
  }
})
```

THE `/did-mount` FOLDER.

SUMMARY

SUMMARY

- YOU DON'T NEED JSX TO WORK WITH REACT.JS. BUT ITS THE RECOMMENDED SYNTAX FOR REACT.JS COMPONENTS.
- JSXTRANSFORMER FOR RUN-TIME JSX (DEVELOPMENT ONLY).
- REACT.JS CAN BE INSTALLED VIA MULTIPLE SOURCES: NPM, WEBSITE, AND CDN.
- JSX TYPE IS `text/jsx`: `<script type="text/jsx">`

SUMMARY (CONT.)

- YOU CREATE REACT.JS ELEMENTS WITH `<...>` OR `React.createElement` AND RENDER THEM WITH `ReactDOM.render`
- STATES ARE MUTABLE. AND PROPS ARE IMMUTABLE
- USING BABEL CAN WATCH FOR FILE CHANGES WITH `-w` FLAG
- REGULAR VS. CUSTOM COMPONENTS: LOWER-CASE FIRST LETTER

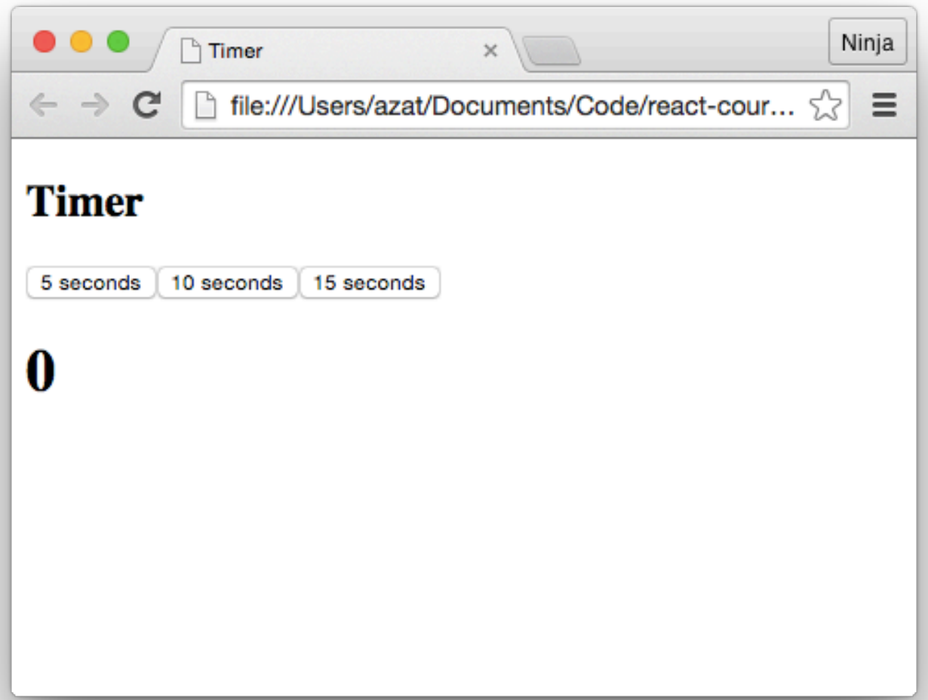
SUMMARY (CONT.)

- `React.createClass` ALLOWS TO CREATE CUSTOM COMPONENTS
- `React.createClass` NEEDS `render` METHOD THAT RETURN OTHER REACT.JS COMPONENT (ALWAYS ONE).

SUMMARY (CONT.)

- FOR AND CLASS ARE FORHTML AND CLASSNAME ATTRIBUTES IN REACT.JS COMPONENTS
- `{}` IS A WAY TO RENDER VARIABLES AND JS IN THE JSX CODE
- `this.state.NAME` AND `this.props.NAME` ARE WAYS TO ACCESS STATE AND PROPS VARIABLES RESPECTIVELY

PROJECT: TIMER



TIMER DEMO

PROJECT: TIMER SOLUTION

HTTP://BIT.LY/1STYTNF

QUESTIONS AND EXERCISES