

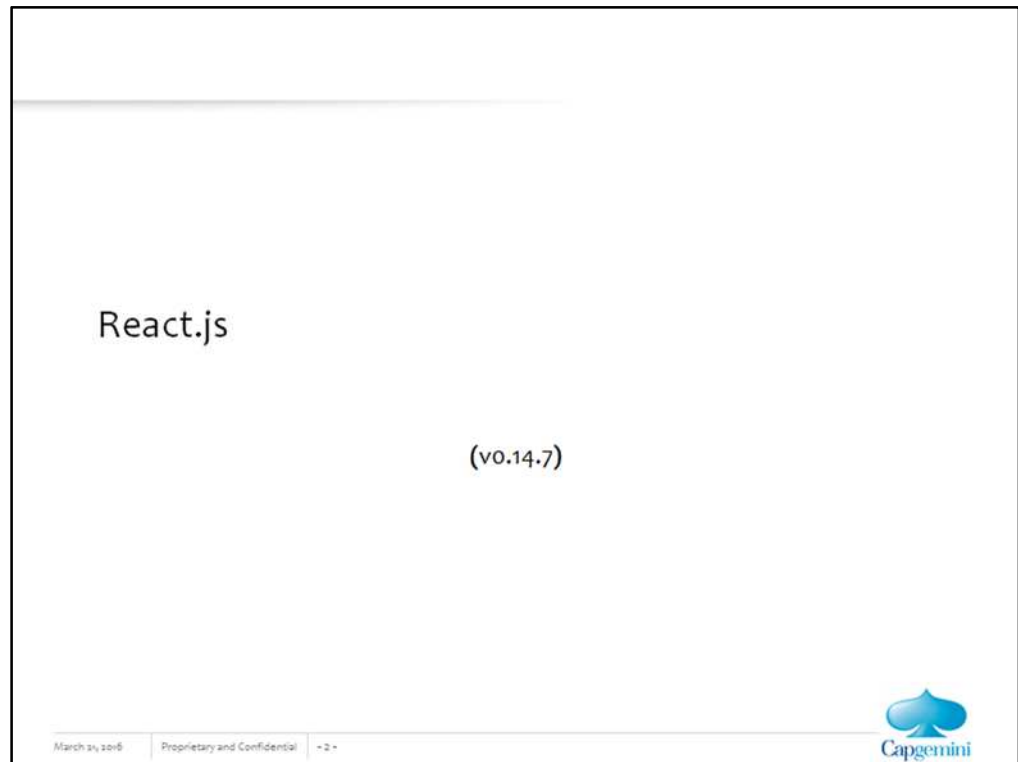
Instructor Notes:



Copyright © 2016 Capgemini. All rights reserved. No part of this publication shall be reproduced in any way, including but not limited to photocopy, photographic, magnetic, or other record, without the prior written permission of Capgemini.

Capgemini considers information included in this document to be Confidential and Proprietary.

Instructor Notes:



Instructor Notes:

Add instructor notes here.

Document History

Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
21/03/2016	1.0	React v0.14.7	Karthik Muthukrishnan	



Instructor Notes:

Add instructor notes here.

Course Goals and Non Goals

➤ Course Goals

- Understand the React Fundamentals using ES5
- Building web applications using React, Node, Flux and Gulp



➤ Course Non Goals

- Comparing React with MV* Framework like AngularJS, Backbone, Ember, etc.
- React using ECMAScript 6 (ES6) next version of JavaScript
- Unit Testing React Components using Jest
- Server-Side Rendering with React, Node, Express and MongoDB

March 24, 2016

Proprietary and Confidential

- 4 -



Instructor Notes:

Add instructor notes here.

Pre-requisites

➤ HTML, JavaScript & CSS

March 24, 2016

Proprietary and Confidential

- 5 -



Instructor Notes:

Add instructor notes here.

Intended Audience

- **Web application developers**



March 24, 2016

Proprietary and Confidential

- 6 -



Instructor Notes:

Add instructor notes here.

Day Wise Schedule

➤ **Day 1**

- Lesson 1: React Introduction
- Lesson 2: The core of React
- Lesson 3: React Fundamentals

➤ **Day 2**

- Lesson 4: Building React Apps with Flux

Instructor Notes:

Add instructor notes here.

Table of Contents

➤ **Lesson 1: React Introduction**

- React Introduction
- Why React?
- Virtual DOM
- How Virtual DOM Works?
- How React Renders the View?
- React Advantages
- React Official Website
- React CDN Hosting

March 24, 2016

Proprietary and Confidential

- 8 -



Instructor Notes:

Add instructor notes here.

Table of Contents

➤ **Lesson 2: React Introduction**

- React Top-Level API
- React.createElement
- ReactDOM.render
- ReactDOMServer.renderToString
- ReactDOMServer.renderToStaticMarkup
- React.createClass
- React.Children.count
- React.Children.map
- React.Children.forEach
- JSX Introduction
- Babel online compiler

March 24, 2016

Proprietary and Confidential

- 9 -



Instructor Notes:

Add instructor notes here.

Table of Contents

➤ Lesson 3: React Fundamentals

- React Element
- React Component
- Working with props(Properties)
- Prop Validation
- Static Methods in React
- Nested Components
- React and CSS
- Adding key for Dynamic Children
- JSX Spread Attributes
- Event Handling
- Working with State

March 24, 2016

Proprietary and Confidential

- 10 -



Instructor Notes:

Add instructor notes here.

Table of Contents

➤ **Lesson 3: React Fundamentals (Contd...)**

- Unidirectional data flow
- Component Types
- props or state?
- React Architecture
- React component's Life Cycle
- React component - Life cycle phases
- React component - Life cycle methods execution sequence
- Mixins

Instructor Notes:

Add instructor notes here.

Table of Contents

➤ **Lesson 4: Building React Apps with Flux**

- Introduction to Node.js
- Modules in Node.js
- Node Package Manager
- EventEmitter
- Creating an EventEmitter
- Gulp-JavaScript Task Runner
- Gulp API
- Creating React Component modules
- Creating gulpfile.js
- React - Routing

March 24, 2016

Proprietary and Confidential

- 12 -



Instructor Notes:

Add instructor notes here.

Table of Contents

➤ **Lesson 4: Building React Apps with Flux (Contd...)**

- Flux Introduction
- Flux - Action
- Flux - Dispatcher
- Flux - Store
- React View (Controller View)
- Flux flow

Instructor Notes:

Add instructor notes here.

References

- Apress Introduction to React by Cory Gackenhaimer
- PACKT Publishing React.js Essentials by Artemij Fedosejev
- <http://facebook.github.io/react>

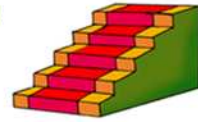


Instructor Notes:

Add instructor notes here.

Next Step Courses (if applicable)

- **Server-Side Rendering with React, Node, Express and MongoDB**



Instructor Notes:

Add instructor notes here.

Other Parallel Technology Areas

- **Angular 2.0**
- **Riot.js - A React-like user interface micro-library**
- **Ractive.js – Template driven UI Library**

March 24, 2016

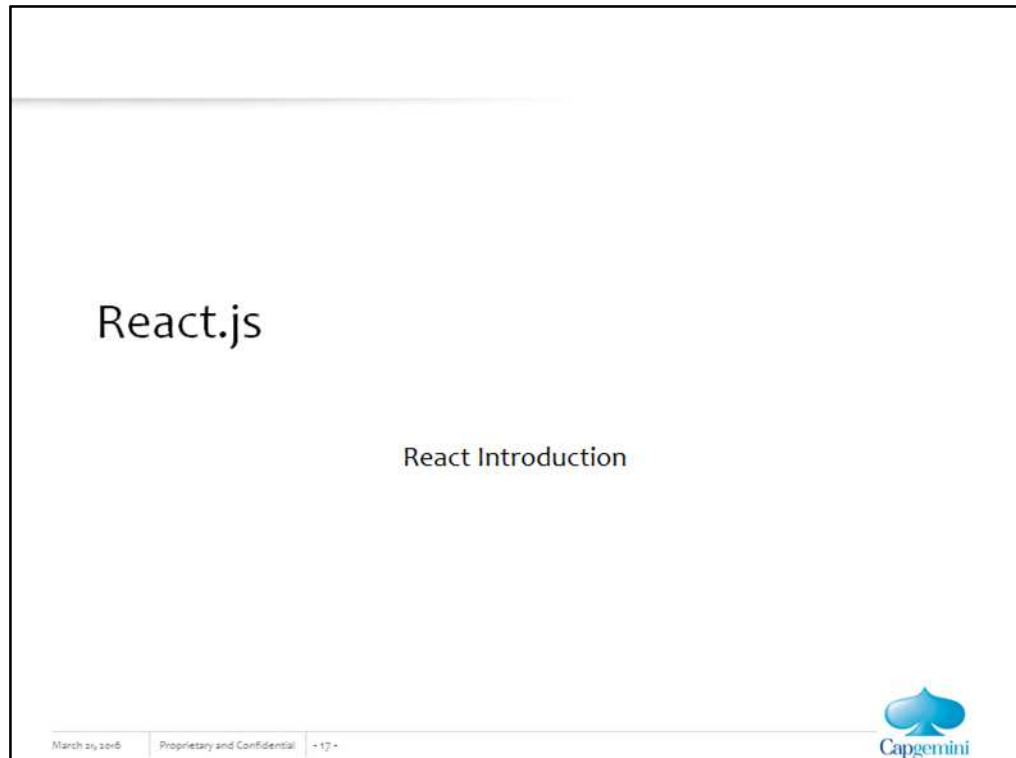
Proprietary and Confidential

- 16 -



Instructor Notes:

Add instructor notes
here.



Instructor Notes:

Add instructor notes here.

Lesson Objectives

- Introduction to React.js
- Understanding Virtual DOM
- React environment setup and using CDN's



Instructor Notes:

Add instructor notes here.

React Introduction


React Introduction

- React is an open source JavaScript library in which components are defined and eventually become HTML.
- It is developed by Facebook and Instagram
- React is intended to be the view("V") or the user interface in MVC.
- One of the benefits and goals of the React project is to make developing a large scale single page application or SPA much easier.

March 24, 2015

Proprietary and Confidential

- 19 -



Instructor Notes:

Add instructor notes here.

React Introduction


Why React?

- Renders views ultra-quickly
- React code is easy to understand for developers, designers, and anyone with the knowledge of XML or HTML.
- Easy to test
- It uses JSX which is clean and easy to understand syntax which can be used directly in our JavaScript.
- Enforces good coding practices
- Supported and used by Facebook

March 24, 2016

Proprietary and Confidential

- 20 -




Instructor Notes:

Add instructor notes here.

React Introduction

Virtual DOM


"React abstract away the DOM from you, giving a simpler programming model and better performance"

- React 

March 24, 2016

Proprietary and Confidential

- 21 -



DOM needs to be manipulated at first, because web applications are not static. Web applications have a state represented by the user interface (UI) that a web browser renders, and that state can be changed when an event (User events and Server events) occurs.

User events: When a user types, clicks, scrolls, resizes, and so on

Server events: When an application receives data or an error from a server, among others.

When we handle the above mentioned events, we update the data that our application depends on

Instructor Notes:

Add instructor notes here.

React Introduction


Virtual DOM

- A DOM(Document Object Model) represents web page in a tree structure. It also refers how these page elements are accessed and changed.
- Updating the DOM is expensive.
- Reading and writing to DOM using DOM API are slow because they are not optimized for speed.
- JavaScript Objects are faster than DOM Objects.
- React offers the Virtual DOM which is a pure JavaScript intermediate representation of DOM.
- React never reads from the real DOM. It only writes to real DOM if needed, so it efficiently handles DOM Updates.
- The process of updating only part of the DOM structure is called as "reconciliation".

March 24, 2016

Proprietary and Confidential

- 22 -



Virtual DOM is nothing but React's local and simplified copy of the HTML DOM. It allows React to do its computations within this abstract world and skip the "real" DOM operations, often slow and browser-specific.

There's no big difference between the "regular" DOM and the virtual DOM. In most cases, HTML code can be changed to a static React component by doing the following:

- Return the HTML code in render
- Replace class attribute name to className - because class is a reserved word in JavaScript.

Three attributes of the virtual DOM that don't appear in "real" DOM are :

1. key,
2. ref
3. dangerouslySetInnerHTML.

Instructor Notes:

Add instructor notes here.

React Introduction

How Virtual DOM Works?

- Whenever the data model state changed, the virtual DOM and React will re-render the UI to a virtual DOM representation.
- React calculates the difference between the two virtual DOM representations: the previous virtual DOM representation that was computed before the data was changed and the current virtual DOM representation that was computed after the data was changed. This difference between the two virtual DOM representations is what actually needs to be changed in the real DOM.
- React updates only what needs to be updated in the real DOM.

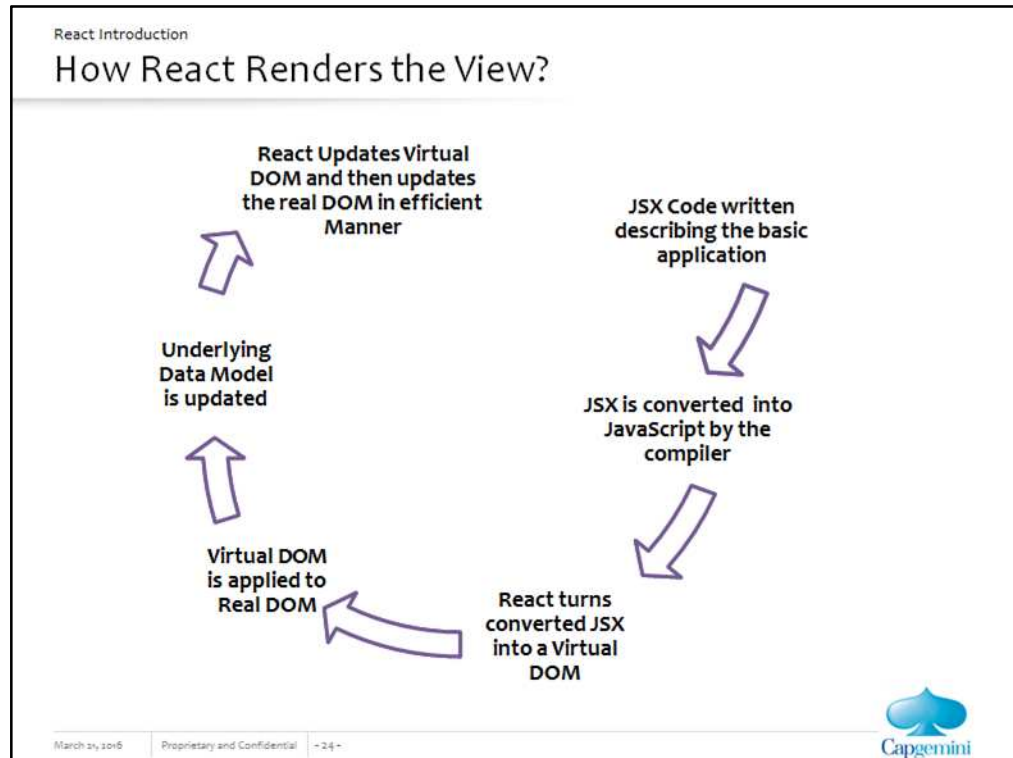
```
graph LR; JSLogic[JS Logic] --> ReactVirtualDOM[React Virtual DOM]; ReactVirtualDOM --> DOM[DOM];
```

The diagram illustrates the interaction between three components: DOM, React Virtual DOM, and JS Logic. JS Logic sends data to the React Virtual DOM, which then updates the real DOM. The React Virtual DOM acts as a bridge between the application logic and the browser's DOM.

March 24, 2016 Proprietary and Confidential - 23 -

Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

React Introduction


React Advantages

- Easy to know how a component is rendered just by look at the render function.
- JSX makes it easy to read the code of components. It is also really easy to see the layout, or how components are plugged/combined with each other.
- React can be rendered on the server-side.
- React is easy to test and integrated with tools like jest
- React ensures readability and makes maintainability easier
- React can be used with any framework like Backbone.js, Angular.js as it is only a view layer

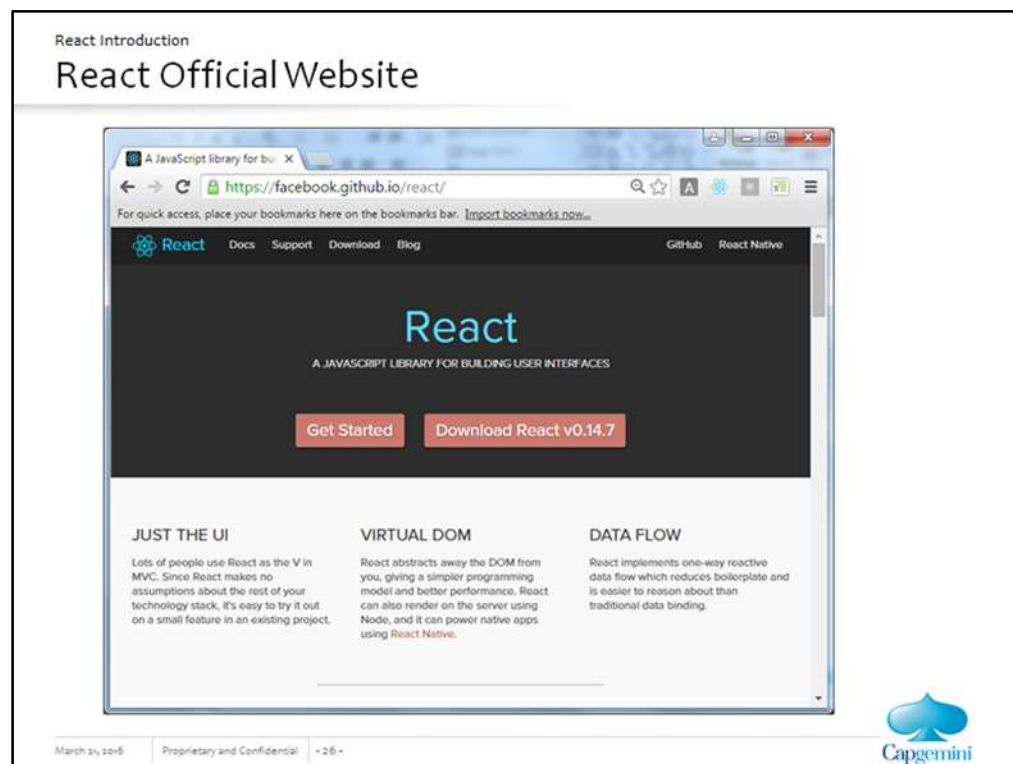
March 24, 2016

Proprietary and Confidential

- 25 -



Instructor Notes:



Instructor Notes:

React Introduction

React CDN Hosting

The screenshot shows the React CDN hosting page. At the top, there's a navigation bar with links like 'cdnjs', 'About', 'Browse Libraries', 'Support us!', 'source', 'git stats', 'Uptime', 'Network', 'Chat', and 'Request a lib'. Below this, the word 'react' is prominently displayed, followed by a link to the GitHub repository. A dropdown menu is set to 'Version 0.14.7'. Below the dropdown, a list of CDN links is provided, including 'react-dom-server.js', 'react-dom-server.min.js', 'react-dom.js', 'react-dom.min.js', 'react-with-addons.js', 'react-with-addons.min.js', 'react.js', and 'react.min.js'. A 'Copy' button is visible next to the 'react-dom-server.min.js' link.

March 24, 2016

Proprietary and Confidential

27

Instructor Notes:

Add instructor notes here.

Summary

- React is a JavaScript Library to build user interfaces for modern web applications.
- React is intended to be the view("V") or the user interface.
- Virtual DOM is nothing but React's local and simplified copy of the HTML DOM.
- Virtual DOM allows React to do its computations within it and skip the "real" DOM operations which is often slow and browser-specific.
- React can be used with any framework as it is only a view layer.
- The process of updating only part of the DOM structure is called as "reconciliation".



March 31, 2016

Proprietary and Confidential

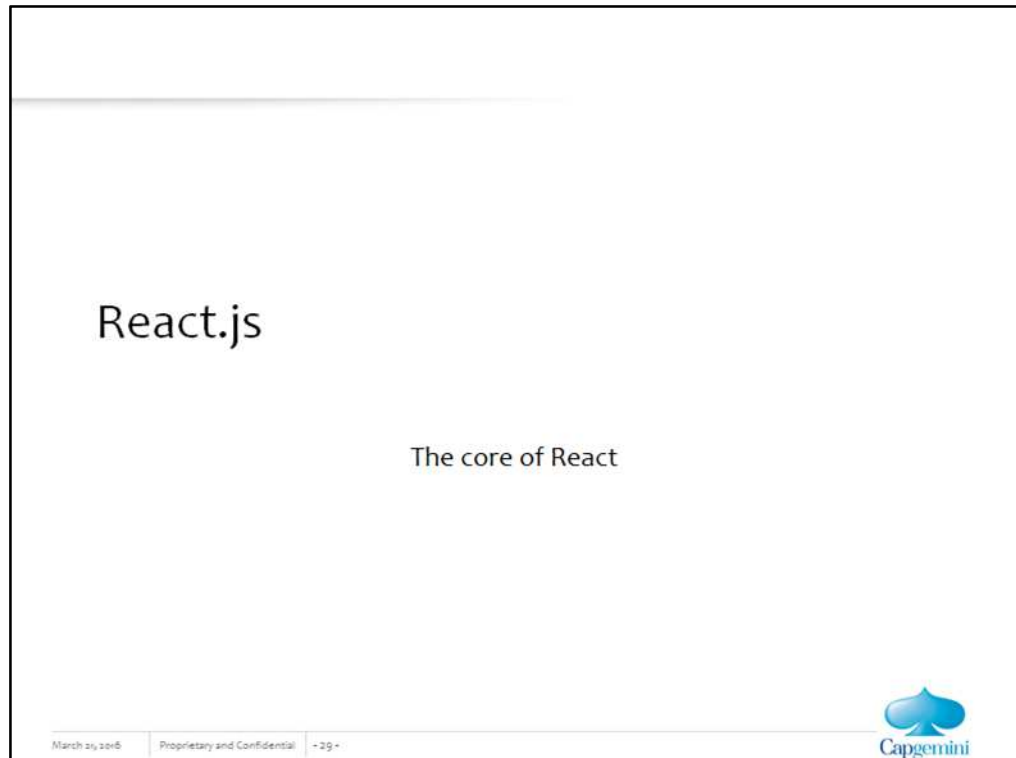
- 25 -



Add the notes here.

Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

Lesson Objectives

- Understanding React' s Top-Level API
- JSX Overview
- Transforming JSX to native JavaScript using Babel compiler



Instructor Notes:

Add instructor notes here.

The core of React


React Top-Level API

- **React is the entry point to the React library.**
- **Following are some of the important functions under React Library**
 - React.createElement
 - ReactDOM.render
 - ReactDOMServer.renderToString
 - ReactDOMServer.renderToStaticMarkup
 - React.createClass
 - React.Children.count
 - React.Children.map
 - React.Children.forEach

March 24, 2016

Proprietary and Confidential

~ 31 ~



Instructor Notes:

Add instructor notes here.

The core of React

React.createElement

- The `createElement` method will generate a new `ReactElement`.
- It is created using at least one, and optionally up to three, arguments to the function: a string type, optionally an object `props`(attributes), and optionally children (text / element).
- `React.createElement(type, [props[, [children ...]]);`

```
var element = React.createElement(  
  'div',  
  { 'data-custom': 'Custom Attribute' },  
  React.createElement('span', null, 'Span-Inner-Text')  
);
```

- **This function available in the following library**
 - <https://cdnjs.cloudflare.com/ajax/libs/react/0.14.7/react.js>

March 24, 2016

Proprietary and Confidential

- 32 -



Instructor Notes:

Add instructor notes here.

The core of React

ReactDOM.render

- Render a `ReactElement` into the DOM in the supplied container and return a reference to the component.
- If the `ReactElement` was previously rendered into container, this will perform an update on it and only mutate the DOM as necessary to reflect the latest React component.
- If the optional callback is provided, it will be executed after the component is rendered or updated.
- `ReactDOM.render(element, container [, callback]);`

```
var element = React.createElement('div', null, 'Hello World');
ReactDOM.render(element, document.getElementById('app'));
<div data-reactid=".1">Hello World</div>
```


- *This function available in the following library*

— <https://cdnjs.cloudflare.com/ajax/libs/react/0.14.7/react-dom.js>

March 24, 2016

Proprietary and Confidential

~ 33 ~



`data-reactid=".1"` attribute in the `div` tag is added and used by React to track the DOM nodes; it might be removed in a future version of React.

Instructor Notes:

Add instructor notes here.

The core of React

ReactDOMServer.renderToString


- Render a `ReactElement` to its initial HTML. This should be used only on the server. React will return an HTML string.
- This method is used to generate HTML on the server and send the markup down on the initial request for faster page loads and to allow search engines to crawl your pages for SEO purposes.
- If `React.render()` called on a node that already has this server-rendered markup, React will preserve it and only attach event handlers to have a very performant first-load experience.
- `ReactDOMServer.renderToString(reactElement);`

```
var element = React.createElement('div', null, 'Hello World');
ReactDOMServer.renderToString(element);
"<div data-reactid=".1" data-react-checksum="337775977">Hello World</div>"
```
- This function available in the following library
 - <https://cdnjs.cloudflare.com/ajax/libs/react/0.14.7/react-dom-server.js>

March 24, 2016

Proprietary and Confidential

~ 34 ~



Instructor Notes:

Add instructor notes here.

The core of React

ReactDOMServer.renderToStaticMarkup

- Similar to `renderToString`, except this doesn't create extra DOM attributes such as `data-react-id`, that React uses internally.
- This is useful if you want to use React as a simple static page generator, as stripping away the extra attributes can save lots of bytes.
- `ReactDOMServer.renderToStaticMarkup(reactElement);`

```
var element = React.createElement('div', null, 'Hello World');
ReactDOMServer.renderToStaticMarkup(element);
"<div>Hello World</div>"
```

- **This function available in the following library**
 - <https://cdnjs.cloudflare.com/ajax/libs/react/0.14.7/react-dom-server.js>

March 24, 2016

Proprietary and Confidential

- 35 -



Instructor Notes:

Add instructor notes here.

The core of React

React.createClass

- The `createClass` method will create a new component class in React.
- `createClass` can be created with an object, which must have a `render()` function
- `React.createClass(specification);`

```
var Hello = React.createClass({
  render: function() {
    return React.createElement("div", null, "Hello ");
  }
});


ReactDOMServer.renderToStaticMarkup(React.createElement(Hello));
"<div>Hello </div>"
```

- **This function available in the following library**
 - <https://cdnjs.cloudflare.com/ajax/libs/react/0.14.7/react.js>

March 24, 2016

Proprietary and Confidential

– 36 –



Instructor Notes:

Add instructor notes here.

The core of React

React.Children.count

- The count method will return the number of components that are contained in element.
- The method accepts a single argument an object.
- `React.Children.count(childrenObject);`


```
var element = React.createElement( "div", null,
  React.createElement( "span", null, "Span-1"
),React.createElement( "span", null, "Span-2" ));

var count = React.Children.count(element.props.children);

console.log(count);
2
```

- **This function available in the following library**
 - <https://cdnjs.cloudflare.com/ajax/libs/react/0.14.7/react.js>

March 24, 2016 Proprietary and Confidential - 37 -



Instructor Notes:

Add instructor notes here.

The core of React

React.Children.map

- It perform a function on each of the immediate children contained and will return an object.
- React.Children.map(children, myFn [, context])**

```
var element = React.createElement( "div", null,
  React.createElement( "span", null, "Span-1"
),React.createElement( "span", null, "Span-2" ));

var childrens =
  React.Children.map(element.props.children,function(child){
    console.log(child);
    return child;
  });
► Object {$$typeof: Symbol(react.element), type: "span", key: null,
ref: null, props: Object...}


► Object {$$typeof: Symbol(react.element), type: "span", key: null,
ref: null, props: Object...}
childrens
[► Object, ► Object]
```

- This function available in the following library**
 - <https://cdnjs.cloudflare.com/ajax/libs/react/0.14.7/react.js>

March 24, 2016

Proprietary and Confidential

- 38 -



Instructor Notes:

Add instructor notes here.

The core of React

React.Children.forEach

- **forEach** is another utility that can be used on element's `props.children` in **React**. It is similar to the `React.Children.map` function except that it does not return an object.
- **`React.Children.forEach(children, myFn [, context])`**

```
var element = React.createElement( "div", null,
  React.createElement( "span", null, "Span-1"
), React.createElement( "span", null, "Span-2" ));
var childrens =
  React.Children.forEach(element.props.children, function(child){
    console.log(child);
  });
```

► `Object { $$typeof: Symbol(react.element), type: "span", key: null, ref: null, props: Object... }`

► `Object { $$typeof: Symbol(react.element), type: "span", key: null, ref: null, props: Object... }`

- **This function available in the following library**
 - <https://cdnjs.cloudflare.com/ajax/libs/react/0.14.7/react.js>

March 24, 2016

Proprietary and Confidential

- 39 -



Instructor Notes:

Add instructor notes here.

The core of React


JSX Introduction

- When we build our virtual DOM by constantly calling the `React.createElement()` method, it becomes quite hard to visually translate these multiple function calls into a hierarchy of HTML tags.
- JSX allows us to create a virtual DOM tree without using the `React.createElement()` method.
- JSX is a JavaScript syntax extension that looks similar to XML.
- JSX can be transformed into native JavaScript using compiler like Babel.
- Babel-core can be downloaded from the following CDN to perform the transformation in the browser
 - <https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.34/browser.js>
- JSX code must be placed within the script tag marked with `type="text/babel"` to convert into JavaScript.

March 24, 2016

Proprietary and Confidential

~ 40 ~



Note : Since JSX is JavaScript, identifiers such as `class` and `for` are discouraged as XML attribute names. Instead, React DOM components expect DOM property names like `className` and `htmlFor`, respectively.

You can't just use HTML comments inside of JSX because it thinks they are real DOM Nodes:

```
render() {
  return (
    <div>
      <!-- This doesn't work! -->
    </div>
  )
}
```

You can use regular `/*` Block Comments `*/`, but they need to be wrapped in curly braces:

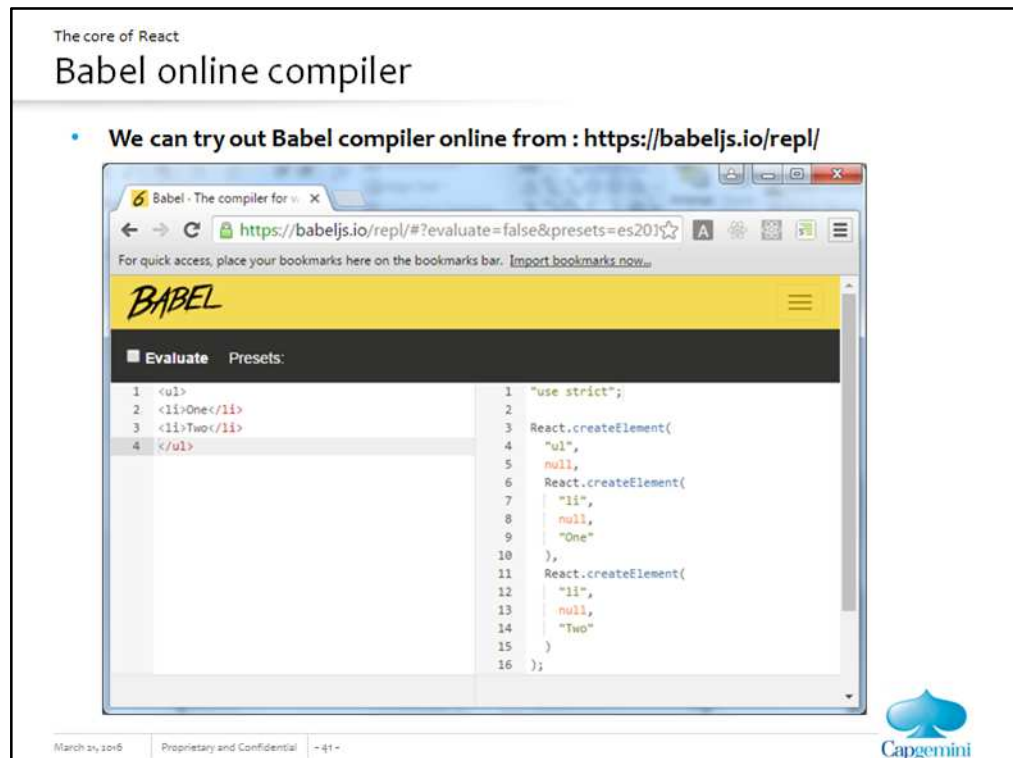
```
{/* A JSX comment */}
```

Same goes for multiline comments:

```
{/*
Multi line
comment
*/}
```


Instructor Notes:

Add instructor notes here.



Babel preprocess the JSX to plain JavaScript at runtime in the browser. Doing this every time slows down the application, but if we transform JSX before the runtime makes the application to run faster. We can transform JSX manually by compiling it through some online compilers like Babel or by installing appropriate node packages in the node environment

```
<html>
  <head>
    <title>REACT - Complex Element</title>
  </head>
  <body>
    <div id="app"></div>
    <script src="./scripts/react.js"></script>
    <script>
      React.render(React.createElement(
        "ul",
        null,
        React.createElement(
          "li",
          null,
          "One"
        ),
        React.createElement(
          "li",
          null,
          "Two"
        )
      ), document.getElementById('app'));
    </script>
  </body>
</html>
```

Instructor Notes:

Add instructor notes here.

Demo

- **Compiling JSX-To-JavaScript-InternalJS**
- **Compiling JSX-To-JavaScript-ExternalJS**



March 25, 2016

Proprietary and Confidential

- 42 -



Instructor Notes:

Add instructor notes here.

Summary

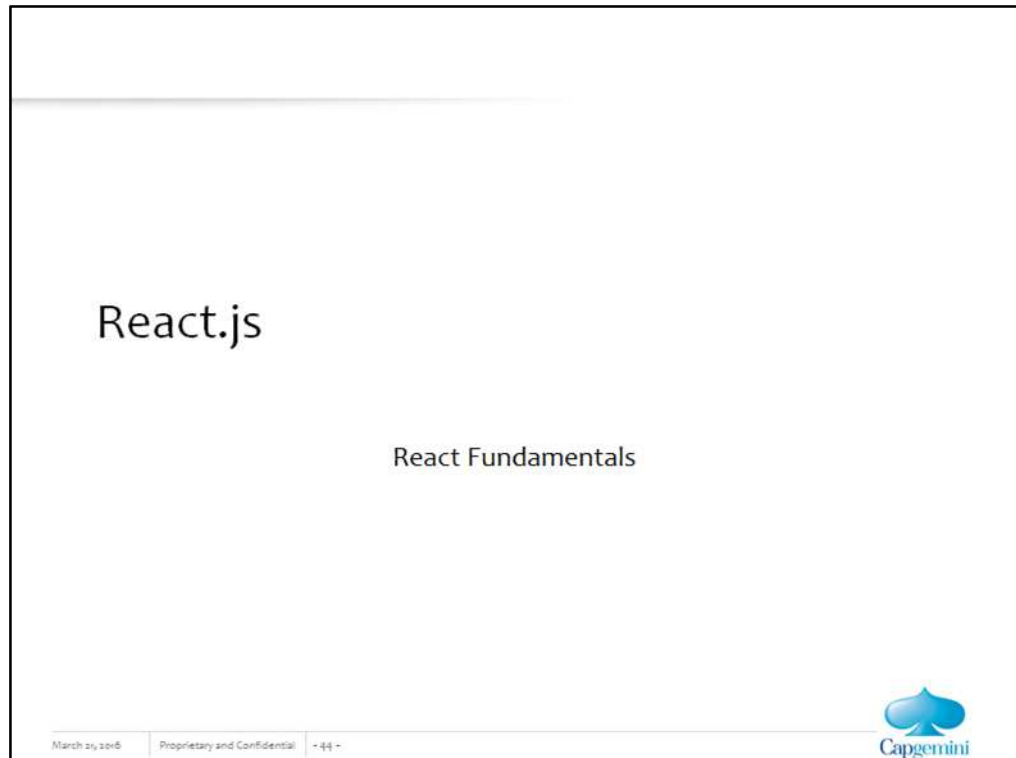
- `React.createClass()` is used to create a new component class based on the given specification. It is available in the package `react`
- `ReactDOM.render()` is used to render a `ReactElement` into the DOM in the supplied container. It is available in the package `react-dom`
- Transforming JSX to native JavaScript before the runtime makes the application to run faster.
- JSX can be manually compiled through online compilers like Babel or by installing appropriate node packages in the node environment.



Add the notes here.

Instructor Notes:

Add instructor notes
here.



Instructor Notes:

Add instructor notes here.

Lesson Objectives

- Understanding React fundamentals
- Understanding React Architecture
- Working with props and state
- Creating composable / nested components
- React component Life cycle
- Creating Mixins



Instructor Notes:

Add instructor notes here.

React Fundamentals


React Element

- A `ReactElement` is a light, stateless, immutable, virtual representation of a `DOM Element`.
- This is the primary type in React.
- `ReactElements` lives in the virtual DOM and makes the basic nodes.
- It's immutability makes it easy and fast to compare and update which results in the React's great performance.
- Almost any HTML tag can be a React Element. JSX compiles HTML tags to `ReactElements`.
- React elements are stateless. Their sole purpose is to construct and render virtual DOM elements.

March 24, 2016

Proprietary and Confidential

- 46 -



Instructor Notes:

Add instructor notes here.

React Fundamentals


React Component

- Every user action triggers a change of a state, but state cannot be added to the stateless React elements, so it needs to be encapsulated in something that already has a state.
- In the React library, React Component encapsulates a component's state and React element which describes how a component need to be rendered.
- New React component can be created using `React.createClass()`
- React component must implement a render function, which instructs the component what to render.
 - Rendering a component means linking it to a DOM element and populating that DOM element.
- React Component can be created with a state(Stateful component) or without a state(Stateless component).

March 24, 2016

Proprietary and Confidential

~ 47 ~



React Components are the building blocks of React.

ReactComponents are great, they are easy to manage. But they don't have access to the virtual DOM where we need to do many things.

Whenever a ReactComponent is changing the state, we need to make those little changes to the “real” DOM. So this is how React deals with it. The ReactComponent is converted to the ReactElement. Now the ReactElement can be inserted to the virtual DOM, compared and updated fast and easily using diff algorithm. It is faster than it would be in the “regular” DOM.

When React knows the diff - it's converted to the low-level (HTML DOM) code, which is executed in the DOM then the code is optimized per browser.

Instructor Notes:

Add instructor notes here.

Demo

- **Creating-React-Component**



March 21, 2016

Proprietary and Confidential

- 48 -



Instructor Notes:

Add instructor notes here.

React Fundamentals

Working with props(Properties)

- The **props** parameter is a JavaScript object(data & event handlers) passed from a parent element to a child element.
- Props are supplied as attributes. If the value of the attribute is JavaScript expression it must be enclosed in curly Brackets (`{}`), if it is a string literal it must be enclosed with in double quotes (`"`)
- Props can be accessed via **props** property inside a component.
- Props are considered to be immutable, i.e. it stores read-only data that is passed from the parent. It belongs to the parent and cannot be changed by its children.
- **`getDefaultProps()`** specifies property values to use, if they are not explicitly supplied.
- Parent can read its children by accessing the special **`this.props.children`** prop.

March 24, 2016

Proprietary and Confidential

- 49 -



```
// Props supplied as attributes
<Sample math={Math.pow(2,3)} name="Veena Deshpande" />

// Access Props via the props property
this.props.math, this.props.name

//SampleComponent has text "Karthik" has its children which can be accessed
via //this.props.children in the component
<SampleComponent>Karthik</SampleComponent>
```

Note: **`this.props.children`** is an opaque data structure, use the **React.Children** utilities to manipulate them.

Instructor Notes:

Add instructor notes here.

Demo

- **Using-Props**



March 21, 2016

Proprietary and Confidential

- 50 -



Instructor Notes:

Add instructor notes here.

React Fundamentals


Prop Validation

- **React offers a great suite of validators for checking the props set for a component are as expected.**
- **React.PropTypes exports a range of validators that can be used to make sure the data you receive is valid.**
- **React supports validation of existence, data type or a custom condition. Using the following prop types we can validate whether a prop :**
 - is required
 - contains a primitive type
 - contains something renderable (a node)
 - is a React Element
 - contains one of several defined types
 - is an array containing only items of a specified type
 - contains an instance of a class
 - contains an object that has a specific shape
- **For performance reasons propTypes is only checked in development mode.**

March 24, 2016

Proprietary and Confidential

~ 51 ~



Prop validations helps us to :

- 1. Immediately see what data a component can process**

propTypes can serve as a sort of mini-reference to your back-end's API by just looking at the code of the component. This eliminates needing to switch between looking at the API documentation and your component code.

- 2. Get console warnings if a component receives an incorrect or missing data type**

If a prop is missing, or has an incorrect data type, you'll see a warning in the JavaScript console. React will only check the propTypes in development mode.

- 3. Check whether API Data is Changed**

It is often the case as a project grows, that the structure of a back-end API response could change, and therefore break an element in the UI if that piece of data is missing, or if a new property is added. Having propTypes can eliminate a whole swath of these kinds of errors. If a new property is added which is not defined in proptype, the console would warn to re-examine the data we're getting from this.props, and update our prop checks accordingly.

- 4. Ensure strong type checking**

Enforcing types in JS is tricky business, but with the proper use of propTypes can really minimize this. prop checks can drastically improve long-term productivity and coerce the code to seem more strongly typed.

Instructor Notes:

Add instructor notes here.

Demo

- **Prop-Validation**



Instructor Notes:

Add instructor notes here.

React Fundamentals

Static Methods in React

- The **statics** object allows you to define static methods that can be called on the component class.
- Methods defined within **statics** block are static, it will be executed before any component instances are created.
- Methods defined within **statics** block do not have access to the props or state of components

```
var MyComponent = React.createClass({  
  statics: {  
    customMethod: function() {  
      ...  
    }  
  },  
  ...  
});  
  
MyComponent.customMethod();
```

March 24, 2016

Proprietary and Confidential

- 53 -




Instructor Notes:

Add instructor notes here.

Demo


- **Static-Methods**



March 21, 2016

Proprietary and Confidential

- 54 -



Instructor Notes:

Add instructor notes here.


React Fundamentals

Nested Components

- **Components can be nested inside other components.**
- **Components are self-contained, so that we can nest any components with another**

```
var OuterComponent = React.createClass({ /*Top level Component*/  
  render:function(){  
    return(<div><InnerComponent/></div>)  
  }  
});  
var InnerComponent = React.createClass(  
  render:function(){  
    return(<div>Inner Component</div>)  
  }  
});  
ReactDOM.render(<OuterComponent/>,document.getElementById('app'));
```

- **React Component which has a child component is called as Top level component.**



March 24, 2016 | Proprietary and Confidential | - 55 -

Top level components can also be called as ControllerView, because it controls the data flow for all of its child component by setting props on children.

Instructor Notes:

Add instructor notes here.

Demo

- **Nested-Components**



March 21, 2016

Proprietary and Confidential

- 56 -



Instructor Notes:

Add instructor notes here.

React Fundamentals

React and CSS


- In React, inline styles are not specified as a string. Instead they are specified with an object whose key is the camelCased version of the style name, and whose value is the style's value.
- When specifying a pixel value for inline style prop, React automatically appends the string "px" after the number value.

```
var divStyle = {height: 10}; // rendered as "height:10px"
ReactDOM.render(<div style={divStyle}>Hello World!</div>, mountNode);
```

March 24, 2016

Proprietary and Confidential

~ 57 ~



List of properties that won't get the automatic "px" suffix:

- animationIterationCount
- boxFlex
- boxFlexGroup
- boxOrdinalGroup
- columnCount
- fillOpacity
- flex
- flexGrow
- flexPositive
- flexShrink
- flexNegative
- flexOrder
- fontWeight
- lineClamp
- lineHeight
- opacity
- order
- orphans
- stopOpacity
- strokeDashoffset
- strokeOpacity
- strokeWidth
- tabSize
- widows
- zIndex
- zoom

Instructor Notes:

Add instructor notes here.

Demo

- **Inline-Styles**



March 21, 2016

Proprietary and Confidential

- 58 -



Instructor Notes:

Add instructor notes here.

React Fundamentals

Adding key for Dynamic Children

- Identity and state of each child must be maintained across render passes.
- Each child in an array or iterator must be uniquely identified by assigning unique key with the help of "key" prop.
- The key should always be supplied directly to the components in the array, not to the container HTML child of each component in the array.
- key is not really about performance, it's more about identity (which in turn leads to better performance).

March 24, 2016

Proprietary and Confidential

- 59 -



Instructor Notes:

Add instructor notes here.

Demo

- Adding-Key-For-Dynamic-Children



March 21, 2016

Proprietary and Confidential

- 60 -



Instructor Notes:

Add instructor notes here.

React Fundamentals

JSX Spread Attributes

- The ... operator is called as spread operator
- Using JSX Spread Attributes, we can construct the props before creating the components and pass them later to the components.

```
var props = {};  
props.foo = x;  
props.bar = y;  
var component = <Component {...props}/>;
```


- The properties of the object that passed in are copied onto the component's props.
- We can transfer it multiple times, combine it with other attributes and override the value.

```
var props = { foo: 'default' };  
var component = <Component {...props} foo={'override'} />;  
console.log(component.props.foo); // 'override'
```

March 24, 2016

Proprietary and Confidential

~ 61 ~



Instructor Notes:

Add instructor notes here.

Demo

- **Transfer-Props**



March 21, 2016

Proprietary and Confidential

- 62 -



Instructor Notes:

Add instructor notes here.


React Fundamentals

Event Handling

- In React, event handlers can be attached to a React element by passing them to the props parameter.
- React uses the **CamelCase** naming convention for event handlers. Ex:- **onClick**

```
var ButtonComponent = React.createClass({
  clickHandler:function(){
    alert('Hi');
  },
  render:function(){
    return (
      <button onClick={this.clickHandler}>{this.props.children}</button>
    );
  }
});

ReactDOM.render(<ButtonComponent>Click</ButtonComponent>,
  document.getElementById('app'));
```



March 24, 2016

Proprietary and Confidential

- 63 -

By default, React triggers the event handlers in the bubble phase, it can be changed to **capturePhase** by appending 'capture' to the event Name : Ex:- **onClickCapture**.

React wraps a browser's native events into the **SyntheticEvent** object to ensure that all the supported events behave identically in Internet Explorer 8 and above.

The **SyntheticEvent** object provides the same API as the native browser's event, which means that you can use the **stopPropagation()** and **preventDefault()** methods as usual. If for some reason, you need to access that native browser's event, then you can do this via the **nativeEvent** property. To enable touch-event handling, simply call **React.initializeTouchEvents(true)**.

Instructor Notes:

Add instructor notes here.

Demo

- **Event-Handling**



March 21, 2016

Proprietary and Confidential

- 64 -



Instructor Notes:

Add instructor notes here.

React Fundamentals


Working with State

- React components can be made dynamic by adding state to it.
- State is used when a component needs to change independently of its parent.
- React component's initial state can be populated using `getInitialState()` with an object map of keys to values.
- React component's state can be accessed using `this.state`
- React component's state can be updated using `this.setState()` with an object map of keys which can be updated with new values. Keys that are not provided are not affected.
- `setState()` merges the new state with the old state.
- As a best practice the nested React components should be stateless. They should receive the state data from their parent components via `this.props` and render that data accordingly.

March 24, 2016

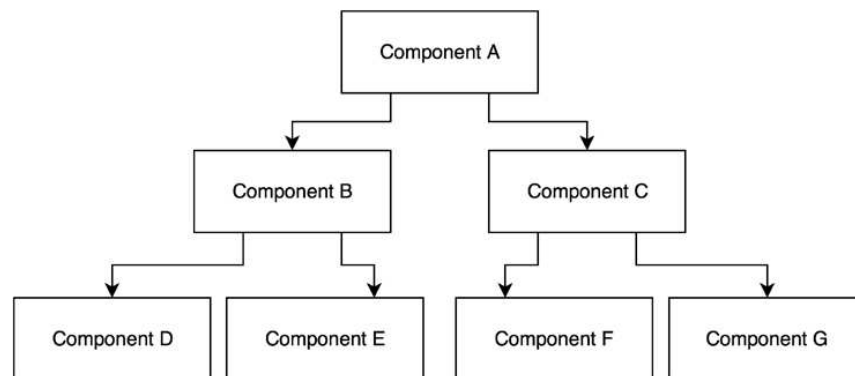
Proprietary and Confidential

- 65 -



Whenever `setState` is called, the virtual DOM re-renders, the diff algorithm runs, and the real DOM is updated with the necessary changes.

React components are composable(Nested components). As a result, we can have a hierarchy of React components. Imagine that we have a parent React component that has two child components, and each of them in turn has another two child components. All the components are stateful and they can manage their own state.




It will be difficult to figure out what the last child component in the hierarchy will render if the top component in the hierarchy updates its state. So as a best practice top-level React component are stateful which encapsulate all of the interaction logic, manage the user interface state, and pass that state down the hierarchy to stateless components, using props.


Instructor Notes:

Add instructor notes here.

Demo

- **Using-State**
- **Using-State-In-Composable-Components**



March 21, 2016Proprietary and Confidential- 66 -

Usage of Function.prototype.bind()

```
this.x = 9;  
var module = {  
  x: 81,  
  getX: function() { return this.x; }  
};
```

```
module.getX(); // 81
```

```
var retrieveX = module.getX;  
retrieveX(); // 9, because in this case, "this" refers to the global object
```

```
// Create a new function with 'this' bound to module  
var boundGetX = retrieveX.bind(module);  
boundGetX(); // 81
```

Instructor Notes:

Add instructor notes here.

React Fundamentals

Unidirectional data flow

- In React, application data flows unidirectionally via the state and props objects, as opposed to the two-way binding of libraries like Angular.
- In a multi component hierarchy, a common parent component will manage the state and pass it down the chain via props.
- state should be updated using the `setState` method to ensure that a UI to update and the resulting values should be passed down to child components using attributes that are accessible in said children via `this.props`

```
graph LR; Root((Root)) -- Render --> ParentChild((Parent/Child)); ParentChild -- Render --> Child((Child)); Child -- Event --> setState(setState); setState -- render --> ParentChild
```

March 24, 2016

Proprietary and Confidential

- 67 -

In two way data binding the view is updated when the state changes, and vice versa. For example, when you change a model in AngularJS the view automatically reflects the changes. Also an input field in the view can alter model. While this works for many apps, it can lead to cascading updates and changing one model may trigger more updates. As the state can be mutated(alter) by both controller and view, sometimes the data flow becomes unpredictable.

React doesn't encourage bi-directional binding to make sure you are following a clean data flow architecture. The major benefit of this approach is that data flows throughout your app in a single direction and you have better control over it.

By keeping the data flow unidirectional you keep a single source of truth. Views are just the functions of the application state. Change in the state will change the view. This is way more predictable and gives a clear idea about how different components react to state change.

Instructor Notes:

Add instructor notes here.

Demo

- **Unidirectional-Flow**



March 21, 2016

Proprietary and Confidential

- 68 -



Instructor Notes:

Add instructor notes here.

React Fundamentals

Component Types

- **Component without state is preferable because state increases complexity and reduces predictability.**
- **In React Components can be divided into**
 - **Stateless Component** : Component only with props, no state. In this type of component besides the render() function all their logic revolves around the props they receive.
 - **Stateful Component** : Those components are also called as state managers. It has both props and state. They are in charge of client-server communication (XHR, web sockets, etc.), processing data and responding to user events.
- **The core logic should be encapsulated in a moderate number of Stateful components, while all visualization and formatting logic should move downstream into as many Stateless Components as possible.**

March 24, 2016


Proprietary and Confidential

- 69 -



Instructor Notes:

Add instructor notes here.

React Fundamentals		
props or state?		
	props	state
Can get initial value from parent Component?	Yes	Yes
Can be changed by parent Component?	Yes	No
Can set default values inside Component?*	Yes	Yes
Can change inside Component?	No	Yes
Can set initial value for child Components?	Yes	Yes
Can change in child Components?	Yes	No
* Note that both <i>props</i> and <i>state</i> initial values received from parents override default values defined inside a Component.		
<div> <div>March 24, 2016</div> <div>Proprietary and Confidential</div> <div>-70-</div> </div> 		

For parent-child communication data need to be passed using props.

Props: props are used to pass data & event handlers down to child components.

- Immutable (let's react do fast reference checks)
- Used to pass data down from your view-controller(top level component)
- Better performance, use this to pass data to child components

State: state is used to store the data which the current page needs in controller-view.

- Should be managed in your view-controller(top level component)
- Mutable
- Worse performance
- Don't access this to from child components, pass it down with props instead

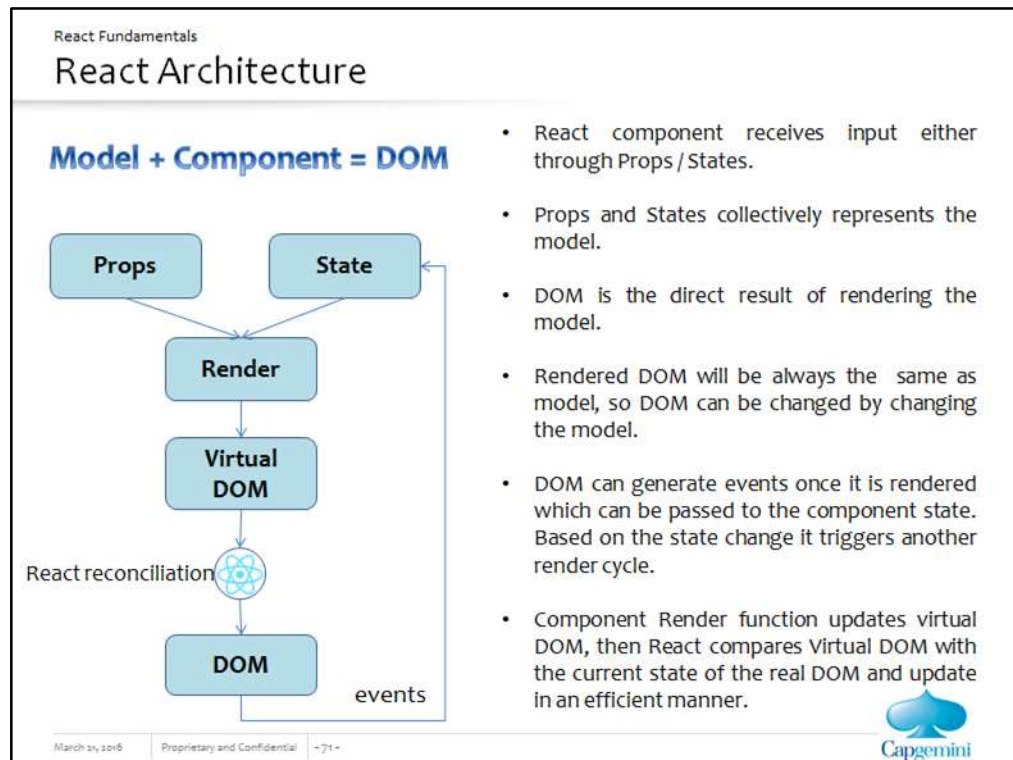
State should contain data that a component's event handlers may change to trigger a UI update. As a best practice create several **stateless** components that just render data, and have a stateful component above them in the hierarchy that passes its state to its children via props. The stateful component encapsulates all of the interaction logic, while the stateless components take care of rendering data in a declarative way

prop or state?

1. Is it passed in from a parent via props? If so, it probably isn't state.
2. Does it change over time? If not, it probably isn't state.
3. Can you compute it based on any other state or props in your component? If so, it's not state.

Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

React Fundamentals

React component's Life Cycle

- **React enables to create components by invoking the `React.createClass()` method, which expects a render method and triggers a lifecycle with certain methods.**
- **It is important to understand component life cycle method role it plays and the order in which it is invoked.**
- **Understanding React component life cycle helps the developer :**
 - To perform certain actions when a new component instance is initialized or destroyed.
 - To write reusable *Mixins* that are invoked at certain points in a component's lifecycle.
 - To perform some calculations when the a parent of a component changes child component's props.
 - To inspect component state/props for better optimization.

March 24, 2016

Proprietary and Confidential

-72-



Instructor Notes:

Add instructor notes here.


React Fundamentals

React Component - Life Cycle Phases

- **React component's Life Cycle has three phases**
 - **Initialization** : Where the component is created
 - **Update** : Where the state and props value changes and triggering updates
 - **Unmounting** : Where the component is deleted
- **The flow of Life cycle methods**

Initialization	Update		UnMounting
Initial Render ↓	Props Change ↓	State Change ↓	Unmount ↓
getDefaultProps*	componentWillReceiveProps	shouldComponentUpdate	componentWillUnmount
getInitialState	shouldComponentUpdate	componentWillUpdate	
componentWillMount	componentWillUpdate	render	
render	render	componentDidUpdate	
componentDidMount	componentDidUpdate		

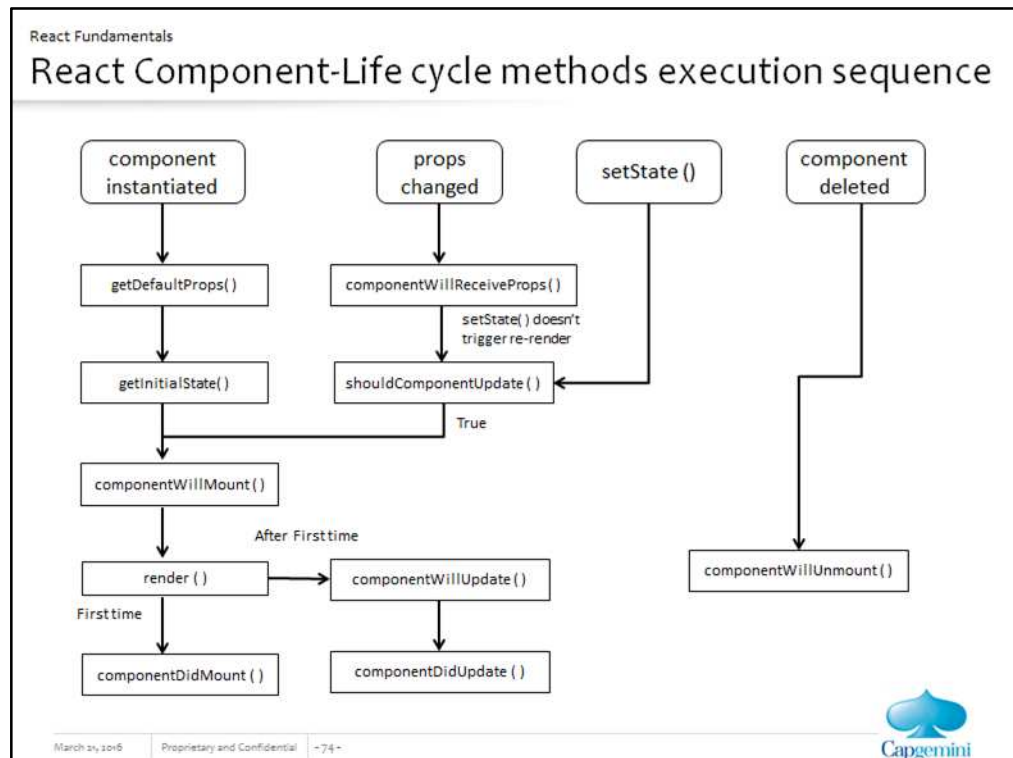
March 24, 2016 Proprietary and Confidential -73-



The invocation of `getDefaultProps` actually takes place once before any instance of the component is created and the return value is shared among all instances of the component

Instructor Notes:

Add instructor notes here.



getInitialState : The object returned by this method sets the initial value of this.state

getDefaultProps: The object returned by this method sets the initial value of this.props. If a complex object is returned, it is shared among all component instances.

componentWillMount : Invoked once immediately before the initial rendering occurs. Calling setState here does not cause a re-render.

render: Returns the jsx markup for a component. Inspects this.state and this.props create the markup. Should never update this.state or this.props

componentDidMount : Invoked once immediately after the initial rendering occurs. We have access the access to get the DOM node using **ReactDOM.findDOMNode()**.

componentWillReceiveProps : Invoked whenever there is a prop change called before render. Not called for the initial render. Previous props can be accessed by this.props. calling setState here does not trigger an additional re-render.

shouldComponentUpdate : Determines if the render method should run in the subsequent step . Called before a render. Not called for the initial render. If the render method to execute in the next step return true, else return false.

componentWillUpdate : Called immediately before a render. this.setState() cannot be used in this method.

componentDidUpdate : Called immediately after a render.

componentWillUnmount : Called immediately before a component is unmounted.

Instructor Notes:

Add instructor notes here.

Demo

- **React-Component-Life-Cycle-Methods-Execution**



March 21, 2016

Proprietary and Confidential

- 75 -



Instructor Notes:

Add instructor notes here.

React Fundamentals

Mixins

- **Mixins are used to share the code between multiple components.**
- **It is an array of objects, each of which can supplement the lifecycle methods.**

```
//To create a React Mixin
var ReactMixin = {
  //Code need to be shared between the components
}


//To use the Mixin in React Component
var FirstComponent = React.createClass({
  mixins:[ReactMixin]
});

var SecondComponent = React.createClass({
  mixins:[ReactMixin]
});
```

March 24, 2016

Proprietary and Confidential

-76-



Instructor Notes:

Add instructor notes here.

Demo

- **Mixins**



March 25, 2016

Proprietary and Confidential

- 77 -



Instructor Notes:

Add instructor notes here.

Summary

- React Components are the building blocks of React.
- props are used to pass data & event handlers down to child components.
- state is used to store the data which the current page needs in controller-view.
- Props and state are related. The state of one component will often become the props of a child component.
- propTypes ensures strong type checking, it can be used to produce console warnings if a component receives an incorrect or missing data type.
- Top level components can also be called as Controller View, because it controls the data flow for all of its child component by setting props on children.



March 31, 2018

Proprietary and Confidential

- 78 -



Add the notes here.

Instructor Notes:

Add instructor notes here.

Summary

- Stateful component are also called as "State Managers"
It has both props and state, where as Stateless component will have only props, no state
- JSX spread attributes are used to merge the old props with additional values, it is used to transfer the props between the components.
- Whenever setState is called, the virtual DOM re-renders, the diff algorithm runs, and the real DOM is updated with the necessary changes.
- React component's Life Cycle has three phases : Initialization, Updation & Unmounting.
- Mixins are used to share the code between multiple components.



March 21, 2018 Proprietary and Confidential - 79 -



Add the notes here.

Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

Lesson Objectives

- Introduction to Node.js
- Working with Modules in Node.js
- Understanding Node's Event Emitter
- Overview of Gulp - JavaScript Task runner
- Creating Gulp tasks
- Routing in React
- Understanding Flux pattern



March 25, 2016

Proprietary and Confidential - 81 -



Instructor Notes:

Add instructor notes here.

Building React Apps with Flux


Introduction to Node.js

- Node.js is a platform built on Chrome's JavaScript runtime(v8 JavaScript Engine) for easily building fast, scalable network applications.
- It is written in C++ and JavaScript.
- JavaScript can be executed in server side using the node platform which promotes JavaScript to run every where. i.e. Server-side and Client-side applications can be created using JavaScript.
- Node adapts JavaScript's non-blocking event loop and make itself has a highly scalable system that uses asynchronous, non-blocking I/O model.

March 24, 2016

Proprietary and Confidential

- 82 -



Node.js CLI

```

D:\Node-Demos>node
> console.log('Welcome to Node.js');
Welcome to Node.js
undefined
> Math.pow(3,2);
9
> console
Console {
  log: [Function: bound ],
  info: [Function: bound ],
  warn: [Function: bound ],
  error: [Function: bound ],
  dir: [Function: bound ],
  time: [Function: bound ],
  timeEnd: [Function: bound ],
  trace: [Function: bound trace],
  assert: [Function: bound ],
  Console: [Function: Console] }
>
<To exit, press ^C again or type .exit>
>

D:\Node-Demos>node --version
v5.9.0

D:\Node-Demos>

```

Instructor Notes:

Add instructor notes here.

Building React Apps with Flux


Modules in Node.js

- **A module encapsulates related code into a single unit of code.**
- **It avoids collision of the methods/variables with other global APIs.**
- **In Node, modules are referenced either by file path or by name.**
- **To work with modules there are three key components :**
 - **require()** : It is used to import the module
 - **exports & module.exports** : It is used to expose any JavaScript objects from one JavaScript file which can be used in other files. Here exports collect all the properties and finally attach them to *module.exports*.

March 24, 2016

Proprietary and Confidential

~ 83 ~



Modules can be referenced depending on which kind of module it is.

Loading a core module

Node has several modules compiled into its binary distribution. These are called the core modules. It is referred solely by the module name, not by the path and are preferentially loaded even if a third-party module exists with the same name.

```
var http = require('http');
```

Loading a file module (User defined module)

We can load non-core modules by providing the absolute path / relative path. Node will automatically adding the .js extension to the module referred.

```
// Absolute path for module.js  
var myModule = require('d:/karthik/nodejs/module');
```

```
// Relative path for module.js (one folder up level)  
var myModule = require('../module');
```

```
// Relative path for module.js (Exists in current directory)  
var myModule = require('./module');
```

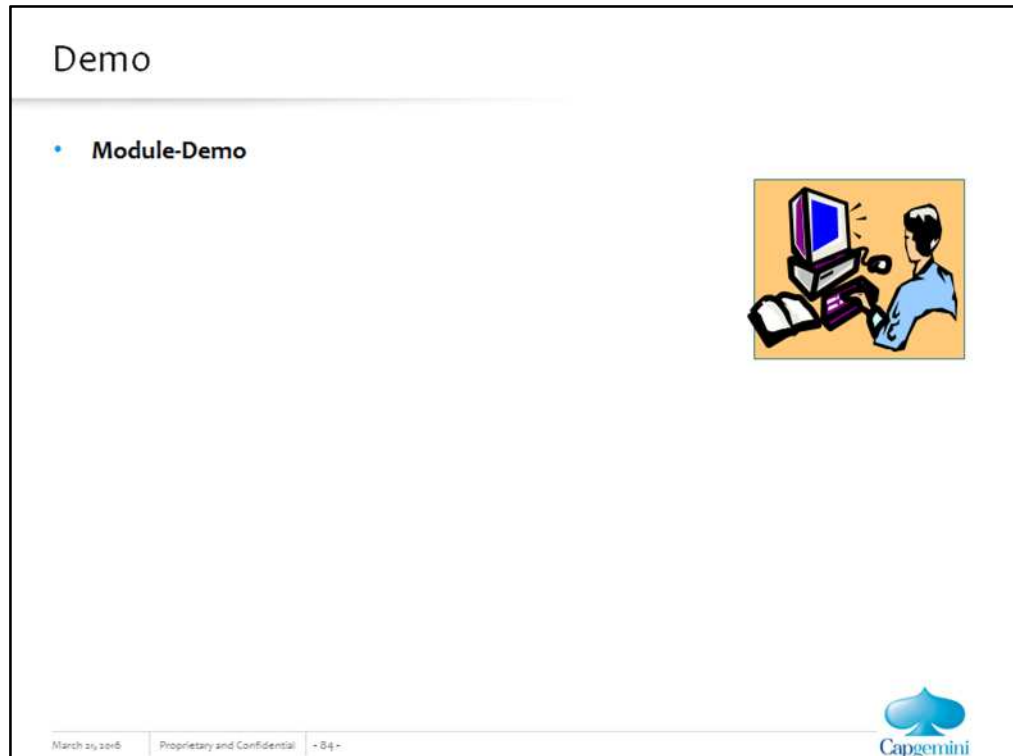
Loading a folder module (User defined module)

We can use the path for a folder to load a module.

```
var myModule = require('./myModuleDir');
```

Instructor Notes:

Add instructor notes here.

**Creating and exporting a module**

- Creating a module that exposes / exports a function called helloWorld.

```
// Save it as myModule.js
exports.helloWorld = function () {
  console.log("Hello World");
}
```

- exports object is a special object created by the Node module system which is returned as the value of the require function when you include that module.
- Consuming the function on the exports object created in myModule.js

```
// Save it as moduleTest.js
var module = require('./myModule');
module.helloWorld();
```

We can replace exports with **module.exports**

```
exports = module.exports = {}
```

To Execute and Test the Module

```
D:\Node-Demos>node moduleTest.js
Hello World
```

Instructor Notes:

Add instructor notes here.

Building React Apps with Flux


Node Package Manager

- Apart from writing our own modules and node core modules, we will frequently use the modules written by other people in the Node community and published on the Internet (npmjs.com).
- We can install those third party modules using the Node Package Manager which is installed by default with the node installation.
 - To install modules via npm use **npm install** command. Ex:- npm install gulp
 - npm installs module packages to the node_modules folder.
- If the module name is not relative and is not a core module, Node will try to find it inside the node_modules folder in the current directory.
 - `var gulp = require('gulp');` //gulp is a third party module

March 24, 2016

Proprietary and Confidential

- 85 -



Loading modules

Modules can be referenced depending on which kind of module it is.

Loading a core module

Node has several modules compiled into its binary distribution. These are called the core modules. It is referred solely by the module name, not by the path and are preferentially loaded even if a third-party module exists with the same name.

```
var http = require('http');
```

Loading a file module (User defined module)

We can load non-core modules by providing the absolute path / relative path. Node will automatically adding the .js extension to the module referred.

```
// Absolute path for module.js
var myModule = require('d:/karthik/nodejs/module');
```

```
// Relative path for module.js (one folder up level)
var myModule = require('../module');
```

```
// Relative path for module.js (Exists in current directory)
var myModule = require('./module');
```

Loading a folder module (User defined module)

We can use the path for a folder to load a module.

```
var myModule = require('./myModuleDir');
```

Instructor Notes:

Add instructor notes here.


Building React Apps with Flux

EventEmitter

- In node.js an event can be described simply as a string with a corresponding callback and it can be emitted.
- The on or addListener method allows us to subscribe the callback to the event.
- The emit method "emits" event, which causes the callbacks registered to the event to trigger.

```
var events = require('events');
var eventEmitter = new events.EventEmitter();
var myCallback = function(data) {
  console.log('Got data: ' + data);
};

eventEmitter.on('karthikEvent', myCallback);
var fn = function() {
  eventEmitter.emit('karthikEvent', 'Data from Emitter');
};
fn();
```

March 24, 2016 Proprietary and Confidential - 86 - 

EventEmitter Methods

All objects which emit events in node are instances of events.EventEmitter which is available inside Event module.

We can access the Event module using require("events")

addListener(event, listener) / on(event, listener)

Adds a listener to the end of the listeners array for the specified event. Where listener is a function which needs to be executed when an event is emitted.

once(event, listener)

Adds a one time listener for the event. This listener is invoked only the next time the event is fired, after which it is removed.

removeListener(event, listener)

Remove a listener from the listener array for the specified event

removeAllListeners([event])

Removes all listeners, or those of the specified event

Instructor Notes:

Add instructor notes here.

Building React Apps with Flux

Creating an EventEmitter

- We can create Event Emitter pattern by creating a constructor function / pseudo-class and inheriting from the EventEmitter.

```
var EventEmitter = require('events').EventEmitter,
    util = require('util');

var Foo = function(){}


util.inherits(Foo, EventEmitter);

Foo.prototype.someMethod = function(){
  this.emit('customEvent', 'Data from Some Method');
}

var fooObj = new Foo();
fooObj.on('customEvent', function(arg){
  console.log('Custom Event Occurred : '+arg);
});

fooObj.someMethod();
```

March 24, 2016 Proprietary and Confidential - 87 -



Instructor Notes:

Add instructor notes here.

Demo

- EventEmitter



March 24, 2016

Proprietary and Confidential

- 88 -



Instructor Notes:

Add instructor notes here.

Building React Apps with Flux


Gulp – JavaScript Task Runner

- Gulp is a JavaScript task runner.
- gulp's use of streams and code-over-configuration makes for a simpler and more intuitive build.
- Gulp is a streaming build system, by using node's streams file manipulation is all done in memory, and a file isn't written until it has been ask to do so.
- Gulp is easy to get installed and running. The steps are:
 - Install Gulp Globally : `npm install -g gulp`
 - Install Gulp In devDependencies : `npm install --save-dev gulp`
 - Optionally install package Gulp-Util log the message in console : `npm install gulp-util`
 - Create a `gulpfile.js`
 - Packages required to work with React are : `gulp-browserify`, `gulp-concat`, `react`, `react-dom`, `reactify`

March 24, 2016

Proprietary and Confidential

- 89 -



gulp-browserify: Browserify lets you require('modules') in the browser by bundling up all of your dependencies.

gulp-concat: Used to concatenate files

react : React is a JavaScript library for building user interfaces. It gives immediate access to React, without requiring the JSX transformer

react-dom : React package for working with the DOM.

reactify : Browserify transform for JSX

Instructor Notes:

Add instructor notes here.

Building React Apps with Flux

Gulp API

- The **gulp api** is incredibly light containing 4 top level functions. They are
 - gulp.task**: It is used to define the tasks.
 - gulp.src**: It is the beginning of the stream which points to the files we want to use. It uses **.pipe** for chaining it's output into other plugins.
 - gulp.dest**: It points to the output folder we want to write files to.
 - gulp.watch**: It allows us to watch files and then performs a task or function.


```

D:\Node-Demos>type NUL > gulpfile.js

/*gulpfile.js*/
var gulp = require('gulp');
var gutil = require('gulp-util');
// create a default task and just log a message
gulp.task('default', function() {
  return gutil.log('Gulp is running!')
});

D:\Node-Demos>gulp
[11:18:55] Using gulpfile D:\Node-Demos\gulpfile.js
[11:18:55] Starting 'default'...
[11:18:55] Gulp is running!
[11:18:55] Finished 'default' after 42 ms
  
```

March 24, 2016 Proprietary and Confidential -90-



gulp.task: It takes three arguments are name, deps and fn. Where name is a string, deps is an array of task names which makes the task given to run once all task mentioned in deps is completed, and fn is the function that performs your task.

```

gulp.task('mytask', function() {
  //do stuff
});

gulp.task('dependenttask', ['mytask'], function() {
  //do stuff after 'mytask' is done.
});
  
```

gulp.src and gulp.dest: mostly used to copy files.

```

gulp.task('copyTxtFiles', function(){
  // copy any txt files in src and its sub folders to public/
  gulp.src('./src/**/*.txt').pipe(gulp.dest('public'));
});
  
```

gulp.watch: To watch files.

```

/*watch for text files in any sbu folders of src for changes*/
gulp.task('watchTxtFiles',function(){
  gulp.watch('./src/**/*.txt',function(event){
    gutil.log('path: '+event.path+'!'+event.type);
  });
});
  
```

Instructor Notes:

Add instructor notes here.

Building React Apps with Flux

Creating React Component modules

app.js

Gulp-Demo

node_modules

src

js

components

app.js

main.js

index.html

gulpfile.js

```
var React = require('react');
var ReactDOM = require('react-dom');

var App = React.createClass({
  render: function() {
    return <h1>React with Gulp</h1>
  }
});

module.exports = App;
```

main.js

Gulp-Demo

node_modules

src

js

components

app.js

main.js

index.html

gulpfile.js

```
var App = require('./components/app');
var React = require('react');
var ReactDOM = require('react-dom');

ReactDOM.render(
  <App />, document.getElementById('main')
);
```

index.html

Gulp-Demo

node_modules

src

js

components

app.js

main.js

index.html


gulpfile.js

```
<!DOCTYPE html>
<html>
  <head>
    <title>React-Gulp</title>
  </head>
  <body>
    <div id="main"></div>
    <script src="js/main.js"></script>
  </body>
</html>
```

March 24, 2016

Proprietary and Confidential

~ 91 ~



Instructor Notes:

Add instructor notes here.

Building React Apps with Flux

Creating gulpfile.js

Gulp-Demo ▾
 > node_modules
 > src
 gulpfile.js

```
var gulp = require('gulp'),
    browserify = require('gulp-browserify'),
    uglify = require('gulp-uglify'),
    connect = require('gulp-connect');

gulp.task('browserify', function() {
  gulp.src('src/js/main.js')
    .pipe(browserify({transform: 'reactify'}))
    .pipe(uglify())
    .pipe(gulp.dest('dist/js'));
});

gulp.task('copyHtml', function() {
  gulp.src('src/index.html')
    .pipe(gulp.dest('dist'));
});

gulp.task('connect', function() {
  connect.server({
    root: 'dist'
  });
});

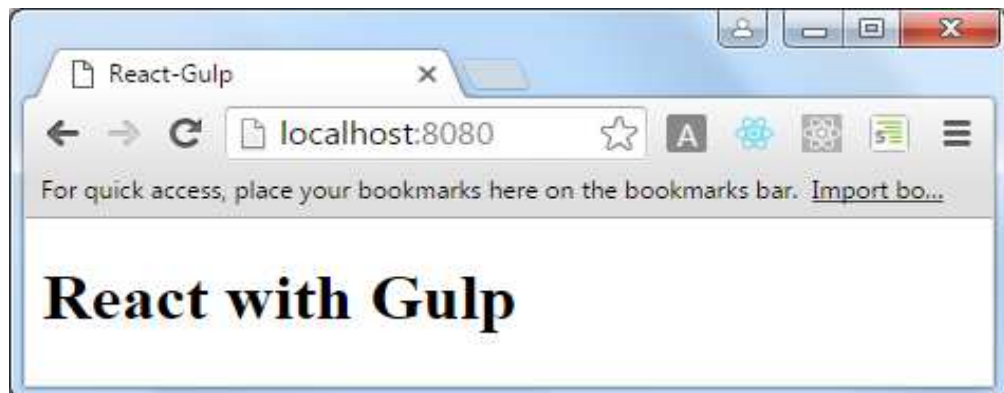
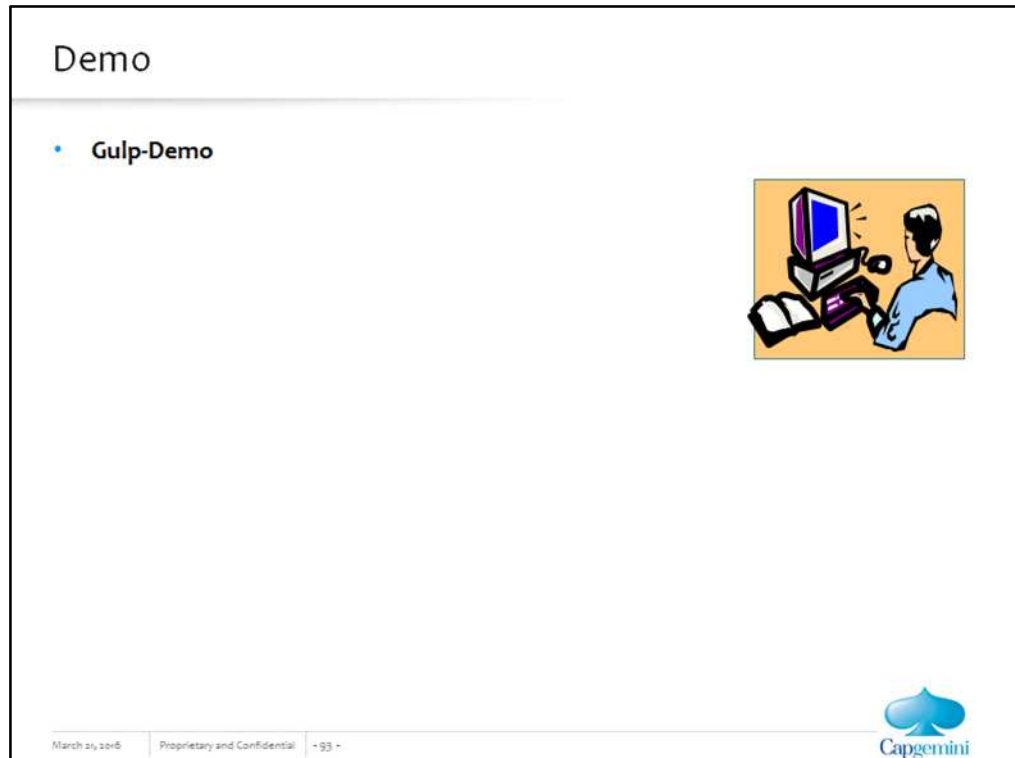
gulp.task('default', ['browserify', 'copyHtml', 'connect']);
```

March 24, 2016 | Proprietary and Confidential | - 92 -



Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

Building React Apps with Flux

Flux Introduction

- Most of the state will go to the top-level component and takes data using `setData` function and pass it down in the tree as props which makes root component tends to grow in terms of code and complexity.
- Flux provides the solution for the above mentioned issue through its four major components : *Stores, Dispatchers, Views/ Components, and Actions.*
- Flux is not a framework, it is a pattern which supports unidirectional data flow.

```
graph LR; A1[Action] --> D[Dispatcher]; D --> S[Store]; S --> RV[React View]; RV --> A2[Action]; A2 --> D;
```

March 24, 2016 Proprietary and Confidential - 94 -

Stores are used to keep data any component can register to the store. When data in the store updates, all registered components gets new version of data from the store. Components gets data from stores and emits actions, which updates stores. Dispatcher dispatch Actions.

Instructor Notes:

Add instructor notes here.

Building React Apps with Flux


Flux - Action

- **Actions (payload) contain no functionality, but rather it describe an event in the application.**
- **Action encapsulates events that occur with in the application like Add User, Delete Item etc.**
- **Actions can be triggered in two different ways:**
 - When user interact with user interface, view will call appropriate actions
 - From the server such as page load or when the error occurred during calls to the server.
- **Payload is typically synonymous with an action. Payload has type and data.**
 - `{ type : CREATE_USER, user : {id : 1, name : 'AnilPatil' } }`

March 24, 2016

Proprietary and Confidential

~ 95 ~



Instructor Notes:

Add instructor notes here.

Building React Apps with Flux


Flux - Dispatcher

- The dispatcher is a singleton, and operates as the central hub of data flow in a Flux application.
- Dispatchers are essentially a registry of callbacks, which can invoke these callbacks in order.
- Dispatchers receive actions and dispatch it to its listeners.
- Dispatcher sends actions to the store.
- An application can have only one dispatcher.
- Dispatcher sends actions to the store.

March 24, 2016

Proprietary and Confidential

~ 96 ~



A dispatcher emits events (`.dispatch()`) to its listeners (`.register(fn)`).

To ensure proper order of execution with multiple listeners, use `.waitFor()` to ensure one is fired after another.

Instructor Notes:

Add instructor notes here.

Building React Apps with Flux

Flux - Store

- **Store are the place where the application data is stored.**
- **Store hold application state, logic and data retrieval methods**
- **Store are not a model, in fact it contains model.**
- **An application can have one or many stores.**
- **Store register callbacks with the dispatcher.**
 - As a best practice only stores should register dispatcher callbacks, react components should never register callback with the dispatcher directly.
- **Stores has no direct setter method, they only accept updates via callbacks**
- **Stores emits changes using Node's EventEmitter.**
- **In flux application stores are the only stakeholders knows how to update data.**
- **Every store has the following common interface**
 - Extends Event Emitter, addChangeListener / removeChangeListener and emitChange

March 24, 2016

Proprietary and Confidential

~ 97 ~



Instructor Notes:

Add instructor notes here.

Building React Apps with Flux

React View (Controller View)

- Top level component in React is also called as Controller View because it control the flow of data down to all of its child components.
- Controller Views are used to interact with stores.
- For instance a store emits an update the controller view should receive the update and pass the updated data to its child components.
- As a best practice top level component should interact with the store, holds the data in its state and passes all the necessary data down to its children via *props*

March 24, 2016

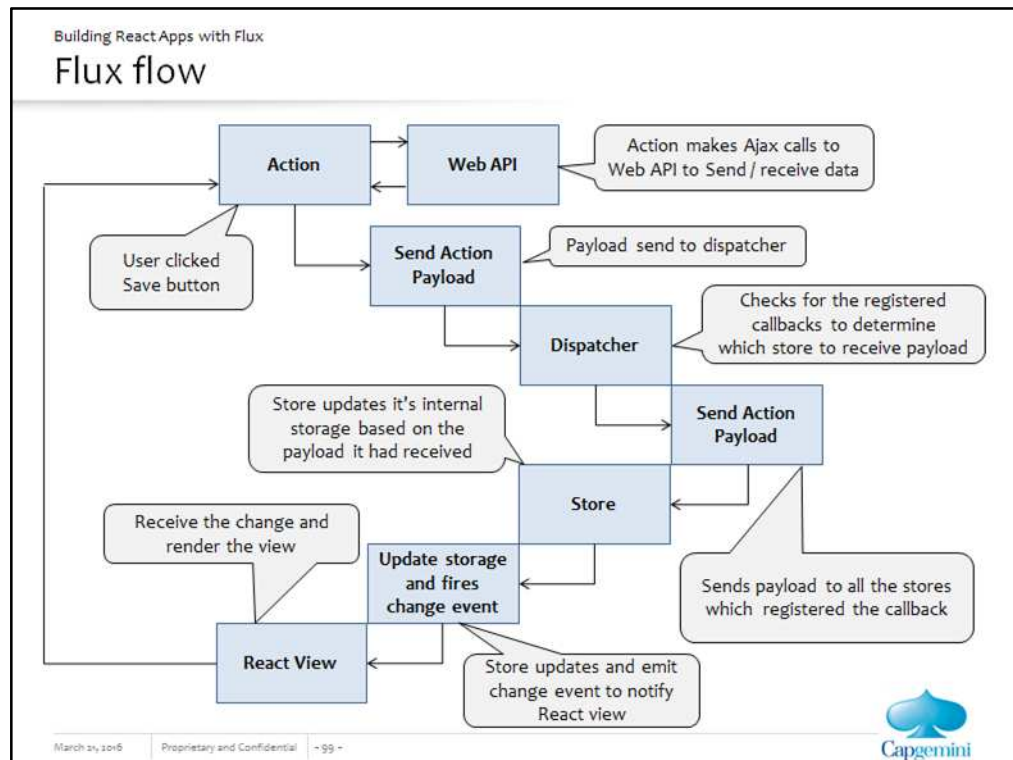
Proprietary and Confidential

- 98 -



Instructor Notes:

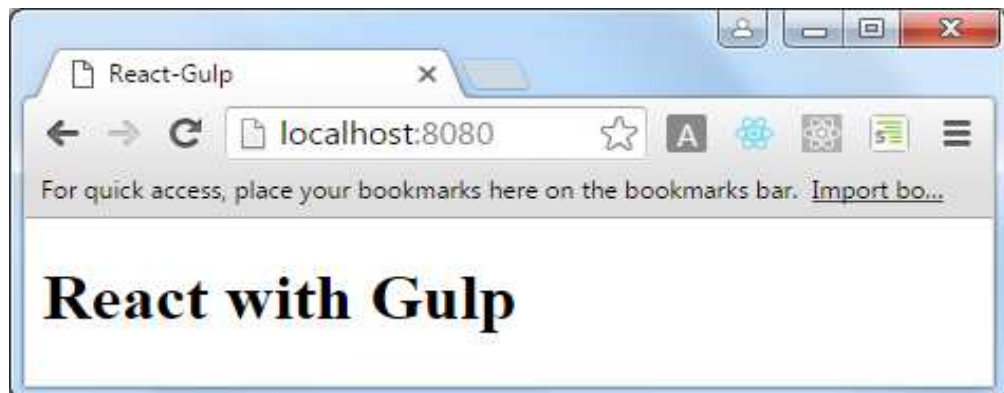
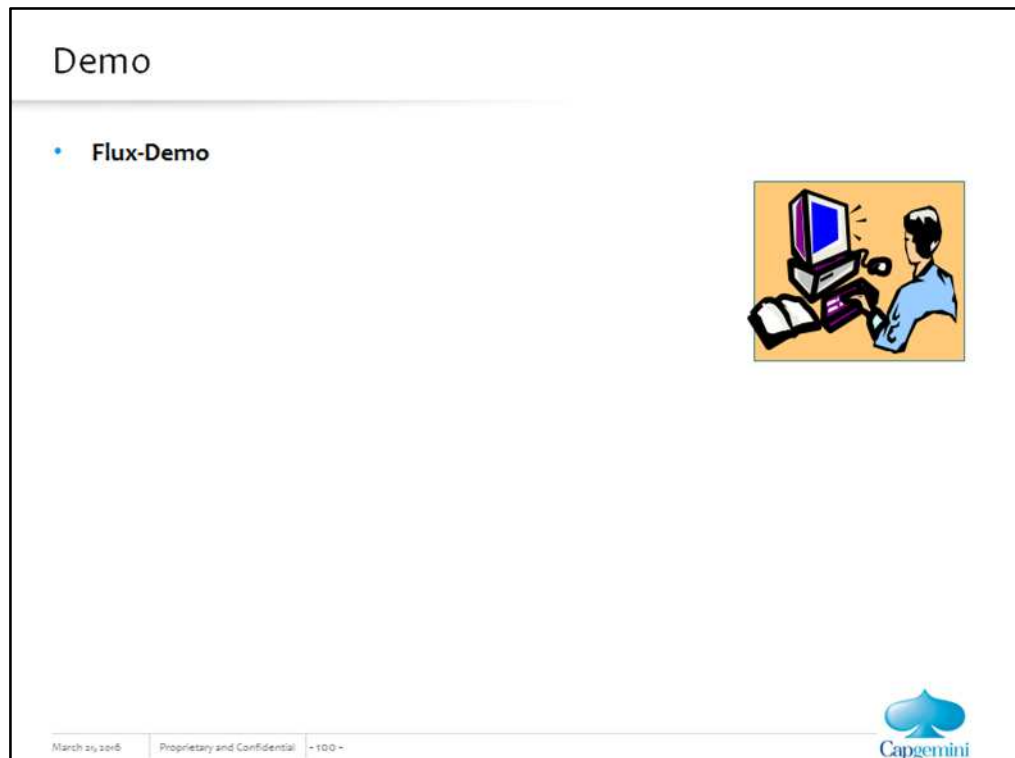
Add instructor notes here.

**Action Flow :**

1. When user click's the Save button in the React component
2. Action register's an action creator with the dispatcher, so that the dispatcher will notify all the registered stores.
3. Dispatcher send the payload to the store based on the callback registration.
4. Store update the payload sent by dispatcher and emit an change event to notify the React Top-level component.
5. React ControllerView update the UI with the new data from the store

Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

Summary

- JavaScript can be executed in server side using the node platform which promotes JavaScript to run every where.
- A module encapsulates related code into a single unit of code
- `module.exports` is used to expose any JavaScript objects from one JavaScript file to another one.
- We can access the module using `require("modulefile")`
- All objects which emit events in node are instances of `events.EventEmitter` which is available inside Event module.
- Gulp is a JavaScript task runner which make use of streams and code-over-configuration.
- Gulp task must be created in a file named "`gulpfile.js`"
- Flux is not a framework it is a *pattern*.



Add the notes here.