Flux & React

Web Application Development

Mark Repka, Rich McNeary, Steve Mueller

Topics Covered

- Introduction to Web Development
  - Get to know some of the basic terms and technologies for general web software
- Flux
  - An architecture for building client-side web applications
- React
  - A JavaScript library for building user interfaces
- Closing notes
  - Main things to remember
- Live Demo
- Questions?

Introduction to Web Development

- HyperText Markup Language (HTML)
  - The standard markup language used to create web pages
  - Consists of tags enclosed in angle brackets: <html>something</html>

- Cascading Style Sheets (CSS)
  - The standard style sheet language used for describing the look and formatting of a document written in a markup language, usually HTML.
  - Uses a JSON-like structure to define classes and attributes for each HTML tag

- JavaScript (JS)
  - Also known as ECMAScript
  - Programming language of HTML and the Web
  - Enables many more dynamic features of websites and web applications

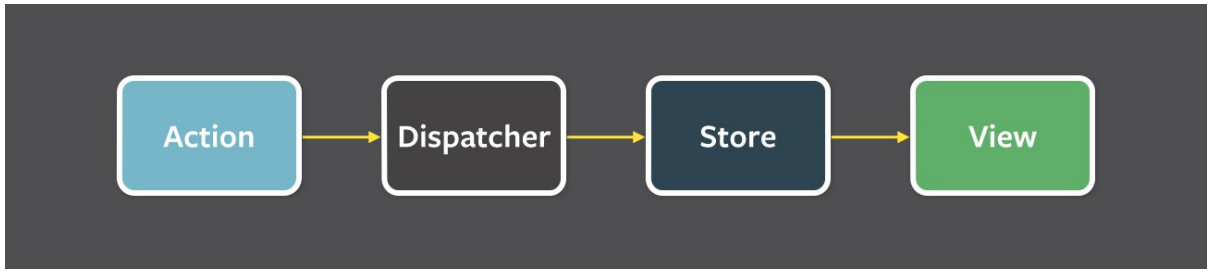
Flux

An architecture for building
client-side web applications

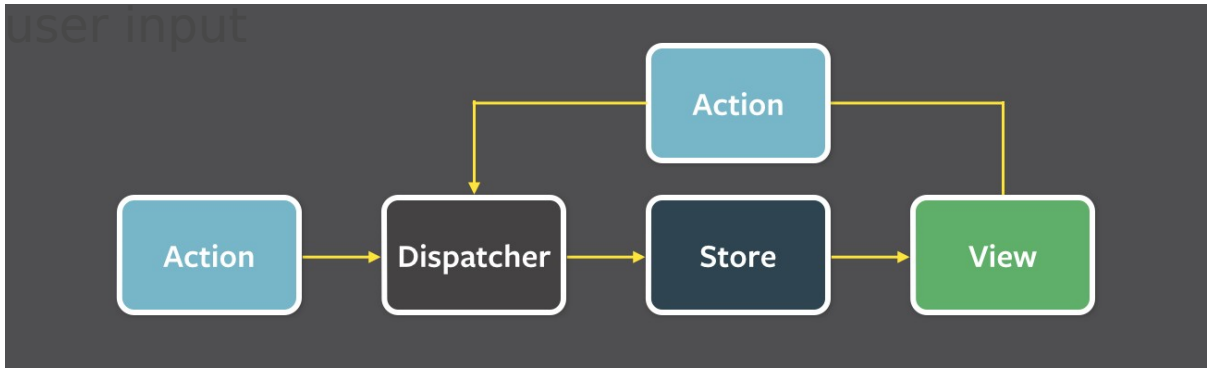
Flux Overview

- NOT a code library.
  - This is just a design pattern
- Flux applications have three major parts
  - Dispatcher
  - Stores
  - Views (React Components)
- Structure and Data Flow
  - Unidirectional data flow is central to the Flux pattern
  - The dispatcher, stores and views are independent nodes with distinct inputs and outputs
- Designed by Facebook and Instagram
- Provided as Open Source software

Data in a Flux application flows in a single direction:



The views may cause a new action response to user input



Flux: Dispatcher

- The dispatcher is the central hub that manages all data flow in a Flux application
- Has no real intelligence of its own
  - It is a simple mechanism for distributing the actions to the stores
- Each store registers itself and provides a callback
- All stores in the application receive the action via the callbacks and can choose to act on them

app.js src

```
1  var React = require('react');
2  var Dispatcher = require('flux').Dispatcher;
3  var store = require('./app/stores/mainstore');
4
5  // Create our dispatcher
6  var dispatcher = new Dispatcher();
7  dispatcher.register(function (action) {
8      store.actionHandler(action);
9  });
10
11 var jsx_view = require('./app/views/TestView');
12 var app = React.createElement(jsx_view);
13 React.render(app, document.getElementById('app'));
14
15
16 // Kick everything off
17 dispatcher.dispatch({ type: "init"});
```

Facebook, the creators of Flux, provide code for an example Dispatcher to get started. This is seen in the require('flux').Dispatcher statement in the code to the right.

Flux: Stores

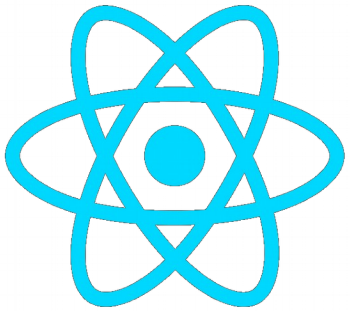
- Stores contain the application state and logic
  - Similar to the model and controller in a traditional MVC application
- Registers itself with the dispatcher and provides it with a callback
- Contains a switch statement that can decide what to do with various actions that are dispatched by the Dispatcher
- Emits events to any listening Views telling them to update their internal state

mainstore.js src\app\stores

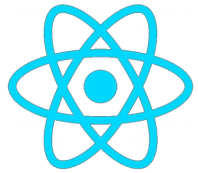
```
1  var EventEmitter = require('events').EventEmitter;
2  var log = require('./logging');
3  var util = require('util');
4
5  function MainStore() {
6      EventEmitter.call(this);
7  }
8
9  util.inherits(MainStore, EventEmitter);
10
11 MainStore.prototype.actionHandler = function (action) {
12     var retval = null;
13     log.log("Got an action: " + action.type);
14     switch (action.type) {
15         case 'init':
16             retval = init();
17             break;
18     }
19     return retval;
20 };
21
22 function init(){
23     log.log("Init!");
24     _store.emit('some_event');
25 }
26
27 var _store = new MainStore();
28 module.exports = _store;
```

Flux: Views

- Flux was designed to pair well with the React library which provides the UI side of your web application
- When an action is captured by the store an event is emitted to any view that is listening to that store.
- The view calls its own setState( ) or forceUpdate( ) methods to update accordingly
  - More on this in the React section next…



React

React 

A JavaScript library for
building user interfaces

React Overview

- Just the User Interface
  - Handles the V in the MVC design pattern
- Simple Components
  - Each react component handles one thing
- Usually written in JSX format
- Allows for XML/HTML-like syntax directly
  - in the JavaScript code
- Translates directly into JavaScript
  - Can be run directly in a standard web browser with no additional libraries required
- Designed by Facebook and Instagram
  - Powers the UI of both of these websites
- Provided as Open Source software

TestView.jsx src\app\views

```jsx
var React = require('react');
var store = require('../stores/mainstore');

module.exports = React.createClass({
    getInitialState: function() {
        return { displayString: "Hello World"};
    },

    componentWillMount: function(){

    },

    componentDidMount: function() {
        store.on('some_event', this.handleEvent);
    },

    componentWillUnmount: function() {

    },

    handleEvent: function() {
        console.log("Got an event, some_event");
        this.setState({displayString: "Hello World, new event!"});
    },

    render: function() {
        return (
            <div>
                <p>{this.state.displayString}</p>
            </div>
        );
    }
});
```

React: Intro to JSX

- JSX is a JavaScript syntax extension that looks similar to XML/HTML
- Offers a concise and familiar syntax for defining components with optional attributes and state
- It's more familiar for casual developers such as designers

```
render: function() {
    return (
        <div>
            <p> This is some JSX here. It looks just like HTML <p>
            <p>{this.state.displayString}</p>
            <p> But we can use JavaScript variables, like above! </p>
        </div>
    );
}
```

Each React component is displayed through its render function. This function returns some JSX that defines how the component will display on the page.

React: Intro to JSX

- This Render function and some helper JavaScript variables lets us do some really useful things!
- Dynamically creating a number of React components from values in a JavaScript array

```
render: function() {
    var urlArray = ['hello world', 'test string!', 'kodak alaris'];
    var reactStrings = urlArray.map(function(string) {
        return (
            <div>
                <p> {string} </p>
            </div>
        )
    });

    return (
        <div>
            <p> This is some JSX here. It looks just like HTML </p>
            <p>{reactStrings}</p>
            <p> But we can use JavaScript variables, like above! </p>
        </div>
    );
}
```

This is some JSX here. It looks just like HTML

hello world

test string!

kodak alaris

But we can use JavaScript variables, like above!

React: Component State

- More complex React components can have some state information that helps them decide how they should render and display.
  - This could include data like our little example array from before
- Some initial state can be defined in a component by overriding the getInitialState( ) function

```
getInitialState: function() {
    return { displayString: "Hello World", someNumber: 1, someArray: ['test', 'test2', 'test3']};
},

componentWillMount: function(){
    // Display one of our initial state values that was defined above:
    console.log(this.state.displayString);
},
```

React: Component State

- If the state changes it will trigger a call to the render method so the component can be updated
- In this simple example, state is just a single integer value called ‘someNumber’ which holds the number of clicks

```
render: function() {
    return (
        <div>
            <button onClick={this.handleClick} > Click Me! </button>
            <p> Number of clicks is {this.state.someNumber} </p>
        </div>
    );
},

handleClick: function() {
    var upCounter = this.state.someNumber + 1;
    this.setState({someNumber: upCounter});
}
```

Web Browser View

React: Component Lifecycle

Components have three main parts of their lifecycle

- **Mounting**: A component is being inserted into the DOM
  - getInitialState( ) is invoked before a component is mounted
  - componentWillMount( ) is invoked immediately before mounting occurs
  - componentDidMount( ) is invoked immediately after mounting occurs
- **Updating**: A component is being re-rendered to determine if the DOM should be updated
  - shouldComponentUpdate( ) is invoked when a component decides whether any changes warrant an update to the DOM
- **Unmounting**: A component is being removed from the DOM
  - componentWillUnmount( ) is invoked immediately before a component is unmounted and destroyed. Cleanup should go here

TestView.jsx src\app\views

```
1  var React = require('react');
2  var store = require('../stores/mainstore');
3
4  module.exports = React.createClass({
5      getInitialState: function() {
6          return { displayString: "Hello World"};
7      },
8
9      componentWillMount: function(){
10
11     },
12
13     componentDidMount: function() {
14         store.on('some_event', this.handleEvent);
15     },
16
17     componentWillUnmount: function() {
18
19     },
20
21     handleEvent: function() {
22         console.log("Got an event, some_event");
23         this.setState({displayString: "Hello World, new event!"});
24     },
25
26     render: function() {
27         return (
28             <div>
29                 <p>{this.state.displayString}</p>
30             </div>
31         );
32     }
33 });
```

React: Virtual DOM

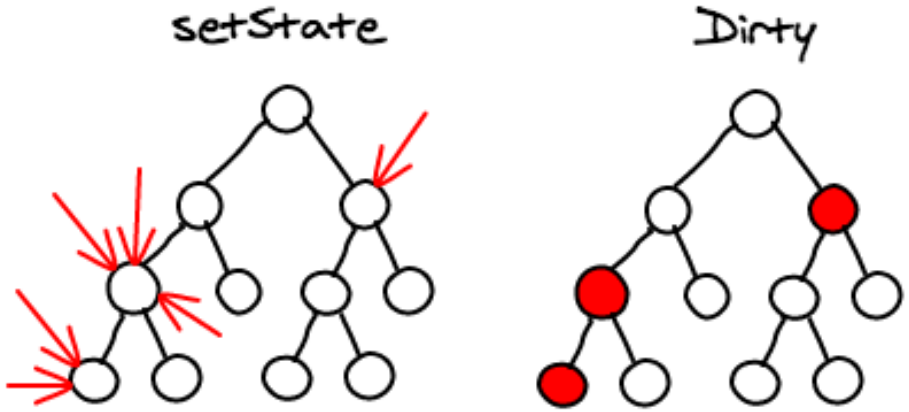
- One of the most important things in any application, web or otherwise, is performance. React has some very interesting features to help improve its performance over other popular web frameworks.

- The Document Object Model (DOM)
  - Structured representation of a document - in this case our web page
  - A tree structure of nodes and objects that each can have different properties and methods

Interacting with this DOM ourselves is slow and tricky to handle correctly. React removes this difficulty of dealing directly with the webpage DOM by introducing the idea of a Virtual DOM

React: Virtual DOM

Whenever you call setState( ) on a component, React will mark it as dirty. At the end of the event loop, when all of the render methods have cascaded through, React looks at all the dirty components and compares the result of the new state to the existing DOM and does a single calculated update.

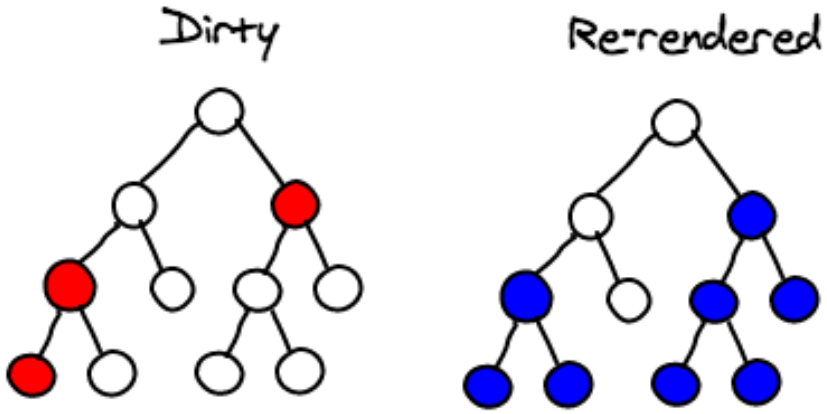


This means that, per any number of updates in that event loop, there is exactly one time when the DOM is being updated.

http://calendar.perfplanet.com/2013/diff/
Facebook Software Engineer, Christopher Chedeau

React: Virtual DOM

This method is great news for performance since we are usually only updating nodes at the bottom of the tree, not at the top!
This means that changes are localized to where the user interacts and does not normally involve updating the entire DOM each time.



http://calendar.perfplanet.com/2013/diff/
Facebook Software Engineer, Christopher Chedeau

React: Helpful Addons

When building an application with React there are some extra features that make development even easier

- Localization with **react-intl**
  - Open Source library provided by Yahoo
  - Uses the standard ICU Message syntax
  - Provides React Components format data and strings

- Animations
  - React provides the **ReactCSSTransitionGroup**
  - Supports basic CSS animations and transitions
  - Wraps all of the components you are interested in animating
  - Specified in CSS file by some name, referenced in code by that name

React: Localization

Web Browser View

The react-intl library, created by Yahoo, provides an easy way to integrate localization into your web application.

React: Animations

- The ReactCSSTransitionGroup addon allows for CSS transitions and animations when a React component enters or leaves the DOM
- This can be seen when the language select dialog opens in the example below
- Animations are then specified by name in the CSS

```
render: function() {
    var dialog = buildDialog(this);
    var containerStyle = this.state.hasDialog ? {} : {display: 'none'};
    var ReactCSSTransitionGroup = React.addons.CSSTransitionGroup;
    return (
        <div id="dialog_container" style={containerStyle}>
            <ReactCSSTransitionGroup component="section" transitionName="dialogAnimation">
                {dialog}
            </ReactCSSTransitionGroup>
        </div>
    );
}
```

Closing: Things to Remember

- React helps you create small simple components can be easily reused throughout the application
- Components can be combined together to create more complex interfaces
- React will automatically manage all UI updates when your underlying data changes
- Page markup and JavaScript behavior are defined together in the same file, the JSX format, making code easy to read and follow
- Designed for performance with large complex web applications
- Cross browser support in Firefox, Chrome, IE8+, Safari, etc.
- Used by industry leaders such as Facebook, Instagram, Netflix, Codecademy

Live Demo!

Lets check out React and Flux in action…

Example Code and Resources

- https://github.com/repkam09/react-flux-test
  - Contains all code shown in this presentation
- http://facebook.github.io/flux/
  - The official documentation for Flux, provided by Facebook.
- https://facebook.github.io/react/
  - The official documentation for React, provided by Facebook.

Any Questions?