React

2/20/2019
CS498RK

MP1 & Takeaways

- CSS is a mess (but SCSS / Sass helps)

- Javascript is a mess (but ES6 helps)

- Javascript is way too forgiving for its own good

Truthy values

```
if (true)
if ({})
if ([])
if (42)
if ("foo")
if (new Date())
if (-42)
if (3.14)
if (-3.14)
if (Infinity)
if (-Infinity)
```

Falsy values

```
if (false)
if (null)
if (undefined)
if (0)
if (NaN)
if ('')
if ("")
if (document.all) [1]
```

What happens when you write messy Javascript?

"Spaghetti code"

Messy,
disorganized code

How to not write spaghetti code?

Modularizing code

ES5

```javascript
(function() {
    // Insert code here
})()
```

```javascript
var module = (function () {
    //private
    return {
    //public
    }
}());
```

ES6

```javascript
module.exports = {
    age: 23,
    getName: () => {
        return "Sujay";
    }
}
```

```javascript
import 'main'
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import

How to not write spaghetti code?

Use strict mode!

- Add at the top of your file

- Catch code bloopers

- Throws errors

- Disables deprecated features

```
"use strict";

// Syntax error
name = "Sujay"
```

How to not write spaghetti code?

Use a linter!

ESLint, JSLint, JSHint check your code for poor practices!

Built-in Editor Support!


```
1    'use strict';
2  ● var foo = "bar";
3         Error  foo is defined but never used (no-unused-vars) at line 2 col 5
4  ● fn(function (err) {});
```

Error foo is defined but never used (no-unused-vars) **at line 2 col 5**
Error Strings must use singlequote. (quotes) **at line 2 col 11**
Error "fn" is not defined. (no-undef) **at line 4 col 1**
Warning Expected error to be handled. (handle-callback-err) **at line 4 col 4**
Error err is defined but never used (no-unused-vars) **at line 4 col 14**

Plain CSS & Javascript still aren't enough...

Awesome Libraries and Preprocessors

Lodash
A modern JavaScript utility library delivering modularity, performance & extras.
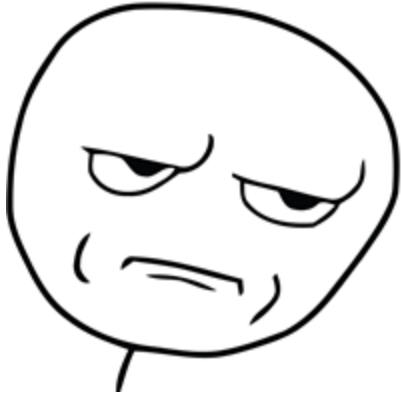
Use ES6 instead of libraries like Lodash! https://github.com/you-dont-need/You-Dont-Need-Lodash-Underscore

If you want to really learn the power of ES6

Still Plain CSS & Javascript still aren't enough...

jQuery

jQuery

- Less overhead & simpler functions

- Fast, small, rich

```
$(document).ready(function(){
    $("p").click(function(){
        $(this).hide();
    });
});
```

- Everything is invoked with a "$"

    - "$" is the name of the function. Seriously.

- Widely-used library for DOM Manipulation

Downsides of jQuery

- Still possible to write "spaghetti code"

- jQuery versioning has become painful

- jQuery doesn't offer a structure, it's just an API

- Growing file size of jQuery can be big overhead on browser load-times

- ES6 has faster native functions

- jQuery hides a lot of "ugly" parts of Javascript, making learning Javascript much more difficult in the long run (http://youmightnotneedjquery.com/)

jQuery has still moved the web forward!

Modern Frameworks & Libraries

How it feels to learn JavaScript in 2016

No JavaScript frameworks were created during the writing of this article.

What do we want?

- More boilerplate

- Access to native functions

- Quick DOM Manipulation

- Enforced Guidelines & Structure

- Build Powerful Web Applications

Angular

- Backed by Google

- Popularized Single-Page Web Applications

- Dynamic Web Application Views

- Services, Factories, Controllers allowed for more modularity

- Directives let you invent your own HTML syntax!

Angular Concepts

Model-View-Controller Paradigm

- Model

    - The data model

- Controller

    - The logic handler

- View

    - Renders the data
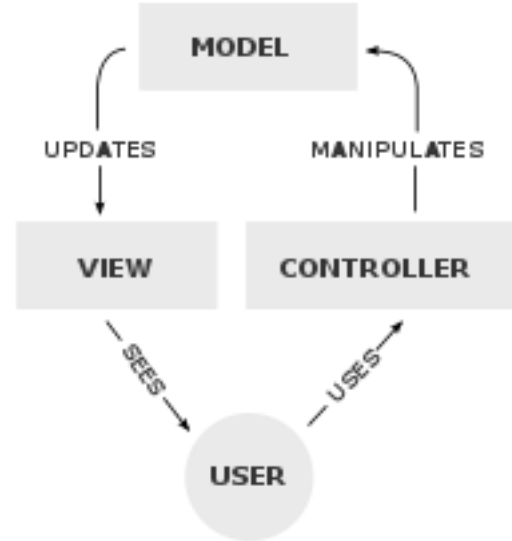
Angular Concepts

Two Way Data-binding

- Automatic synchronization between the model and the view

- Checks for changes in the model or view and does "dirty-checking"

    - $apply calls $digest in Angular, which keeps checking to see if the value has changed!

MVC in Angular

```javascript
// create the module and name it
var myApp = angular.module('myApp', []);

// create the controller and inject Angular's $scope
myApp.controller('MainController', function($scope) {

    // create a message to display in our view
    $scope.message = 'Everyone come and see how good I look!';
});
```

MVC in Angular

{{ }} allows for automatic databinding

```
<div id="main">
    {{ message }}
</div>
```

Why'd we switch?

- Angular 2.0 was completely different from Angular 1.0

- MVC doesn't scale, according to Facebook

    - You end up with a lot of controllers and views to maintain

- React has better performance

    - Angular uses digest cycles & dirty checking

- React leaned more towards ES6 functions rather than custom "directives"

    - Angular introduced ng-*, which was useful, but specific to Angular

- Industry shift towards React

React

- Backed by Facebook

- Library, not Framework

    - For the bigger architecture, look into Flux

- Unidirectional data flow

- Declarative

    - Every component has a state with data injected into it

- Component-based

    - Each module manages its data and views

"React is a library for building composable user interfaces. It encourages the creation of reusable UI components which present data that changes over time."
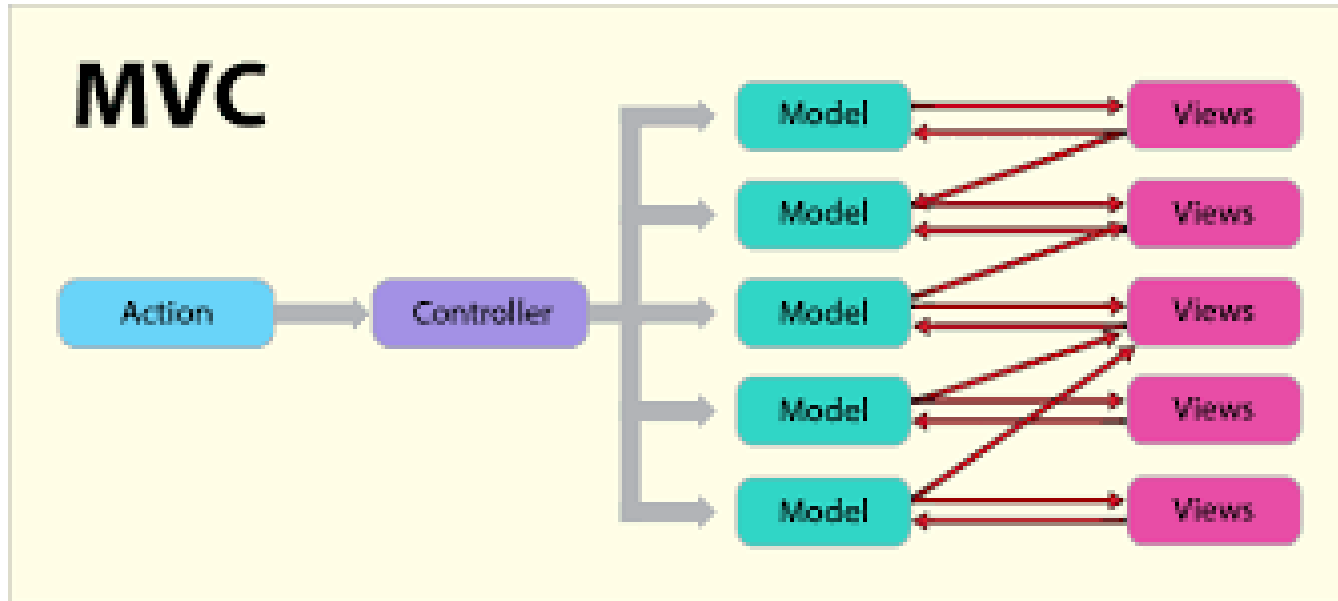
Unidirectional Flow

https://www.youtube.com/watch?v=nYkdrAPrdcw

MVC is too complicated

Why did Facebook think of this?
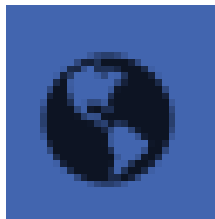
0 Notifications   1 Notification

- Our data has been changed!
- React has noticed this data change and will update the view to reflect this change in data
- How can we display this change without re-rendering the entire page?

A Background on Painting

- The DOM tree is converted into pixels that are laid out onto the page to create the render tree

- "Painting"

- Reflows cause the DOM Tree to be repainted into a newly updated render tree

- Reflow occur due to changes in the DOM tree

    - Updating a DOM Node, adding a DOM node, etc

- Behind-the-scenes computation creates new visual representations

- This can be expensive and slow for our webpage!

    - If only there was a better way!
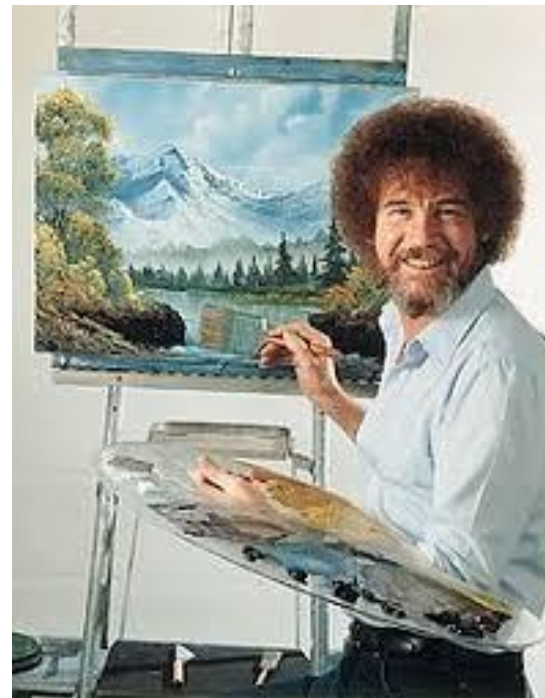
The Joy of (DOM) Painting

Virtual DOM

In-memory, lightweight clone of the DOM, represented as a single JS Object

Repaint the DOM with the smallest amount of changes possible

- First, React notices that the data has changed

- React will execute the change within the lightweight Virtual DOM

- React compares the Virtual DOM with the real DOM by using "diff"

- React immediately patches changes from the Virtual DOM to the real DOM

- This avoids expensive traversing of the DOM tree

Makes React very fast with DOM changes

Thinking in React

Search...

☐ Only show products in stock

Name	Price
Sporting Goods	
Football	$49.99
Baseball	$9.99
Basketball	$29.99
Electronics	
iPod Touch	$99.99
iPhone 5	$399.99
Nexus 7	$199.99

https://reactjs.org/docs/thinking-in-react.html

Thinking in React

Search...

☐ Only show products in stock

Name	Price
Sporting Goods	
Football	$49.99
Baseball	$9.99
Basketball	$29.99
Electronics	
iPod Touch	$99.99
iPhone 5	$399.99
Nexus 7	$199.99

Our Data!

```
[
  {category: "Sporting Goods", price: "$49.99", stocked: true, name: "Football"
  {category: "Sporting Goods", price: "$9.99", stocked: true, name: "Baseball"}
  {category: "Sporting Goods", price: "$29.99", stocked: false, name: "Basketba
  {category: "Electronics", price: "$99.99", stocked: true, name: "iPod Touch"}
  {category: "Electronics", price: "$399.99", stocked: false, name: "iPhone 5"}
  {category: "Electronics", price: "$199.99", stocked: true, name: "Nexus 7"}
];
```

https://reactjs.org/docs/thinking-in-react.html

Thinking in React

https://reactjs.org/docs/thinking-in-react.html

1. **FilterableProductTable (orange):** contains the entirety of the example

2. **SearchBar (blue):** receives all *user input*

3. **ProductTable (green):** displays and filters the *data collection* based on *user input*

4. **ProductCategoryRow (turquoise):** displays a heading for each *category*

5. **ProductRow (red):** displays a row for each *product*

A Simple Example

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

- React Component needs a render method!

- Render returns JSX

- "this.props" is referring to the passed in "props" of the component

JSX

- React's fancy syntax extension to Javascript that looks a lot like HTML

- Creates React "Elements" which allows for injection of Javascript variables and expressions

- Some differences with HTML

    - E.g. "className" vs "class-name"

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

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

Props

All valid React components accept a single "props" object with data and returns a React element

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Props are passed into the React Component & are read-only

```
<Welcome name="Sujay" />
```

https://codepen.io/gaearon/pen/VKQwEo?editors=0010

State

A component's state, or internal data

In the example, name is added to the state. State is an object.

IMPORTANT: State cannot be reassigned, it can only be changed using "this.setState(...)"

this.setState triggers the render function to run again!

```javascript
class Message extends React.Component {
  constructor(props) {
    super(props);
    this.state = {name: ""};
  }

  changeName(newName) {
    this.setState({
        name: newName
    })
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>My name is {this.state.name}.</h2>
      </div>
    );
  }
}
```

http://codepen.io/gaearon/pen/dpdoYR?editors=0010

Lifecycle methods

componentWillMount() - called before the DOM tree has been rendered

componentWillReceiveProps() - called when new props are passed into the component

componentDidMount() - code that will be executed after the component output has been rendered to the DOM

componentWillUnmount() - code that will be run before the webpage is destroyed

React Composable Views

Rendering Multiple Items

- Use regular ES5 and ES6 functions to render JSX or Javascript objects!

- JSX elements are just Javascript objects so you can store them in arrays too!

```javascript
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li>{number}</li>
);
```

React Router

```
  return (
    <Router>
      <Switch>
        <Route exact path="/" component={Home}/>

        {/*
          Add routes for new pages here!
          Here's an example. To view this route, just go to http://localhost:3000/example
        */}
        <Route exact path="/example" component={Example}/>
        <Route exact path="/pokemon" component={Pokemon}/>
      </Switch>
    </Router>
```

https://reacttraining.com/react-router/web/guides/philosophy

Demo

https://github.com/uiuc-web-programming/react-demo

Side Note: Use Classes

The web is littered with examples of React in ES5. Use Classes!

You can convert a functional component to a class in five steps:

1. Create an [ES6 class](#) with the same name that extends React.Component.

2. Add a single empty method to it called render().

3. Move the body of the function into the render() method.

4. Replace props with this.props in the render() body.

5. Delete the remaining empty function declaration.

Further Readings

https://reactjs.org/

https://reactjs.org/blog/2013/06/05/why-react.html

https://reactjs.org/docs/introducing-jsx.html

https://reactjs.org/docs/jsx-in-depth.html