~~From Soup to Sandwich~~

Chalky Soup

Making MVC Java Classes and Front-end Development Work for You

Presented, *with no room AV*,

by Eric McCormick

This ~~Session~~ Is A Chalk Talk

"A ConnectED ChalkTalk is a relaxed, interactive discussion, with no provided media or AV. The purpose of the discussion is to gather input, feedback, and learn from each other."

I am meant to facilitate a discussion. These slides will further as a supplementary aid, just not a primary visual aid.

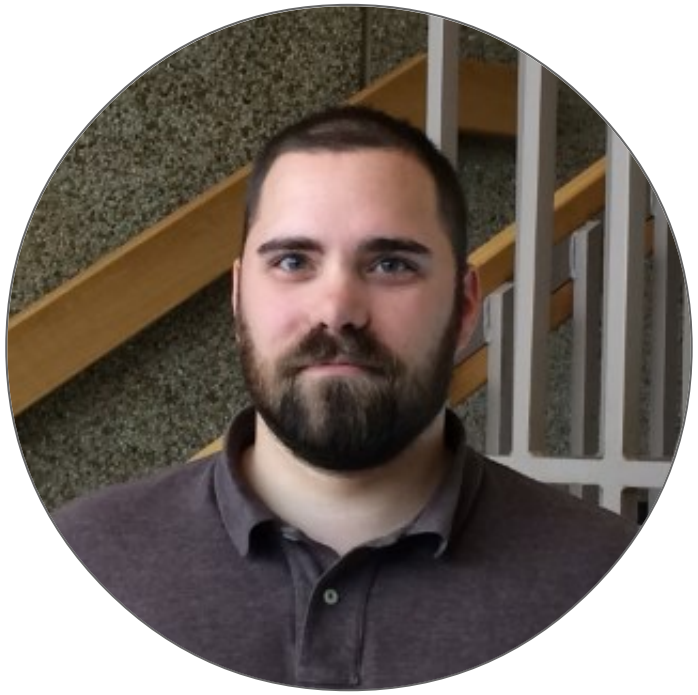
This Chalk Talk

Format for this Chalk Talk (7am-7:45am):

Sequence	Speaker	Time
Introduction	Eric	3 min
Topics to Cover	Eric	2 min
1: Java M-V-C	Eric	5 min
Discussion	Group	5 min
2: Bootstrap(CSS)	Eric	5 min
Discussion	Group	5 min
3: AngularJS	Eric	5 min
Discussion	Group	5 min
Wrap Up, Q&A	Eric/Group	10 min

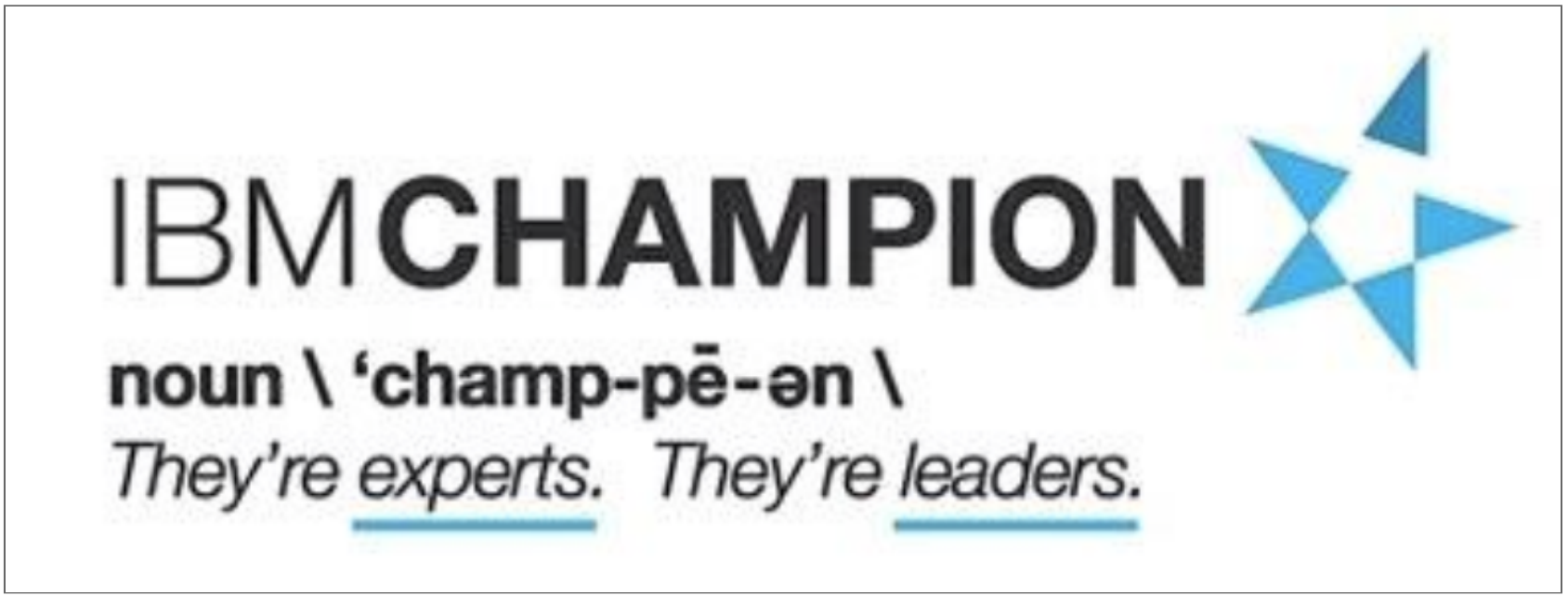
Who Am I?

Eric McCormick



Twitter: **@edmoose**
Blog: **edmoose.github.io/DevBlog/**
XPages developer since ‘11
Java developer on and off since ‘02

Experimented with WebSphere in college

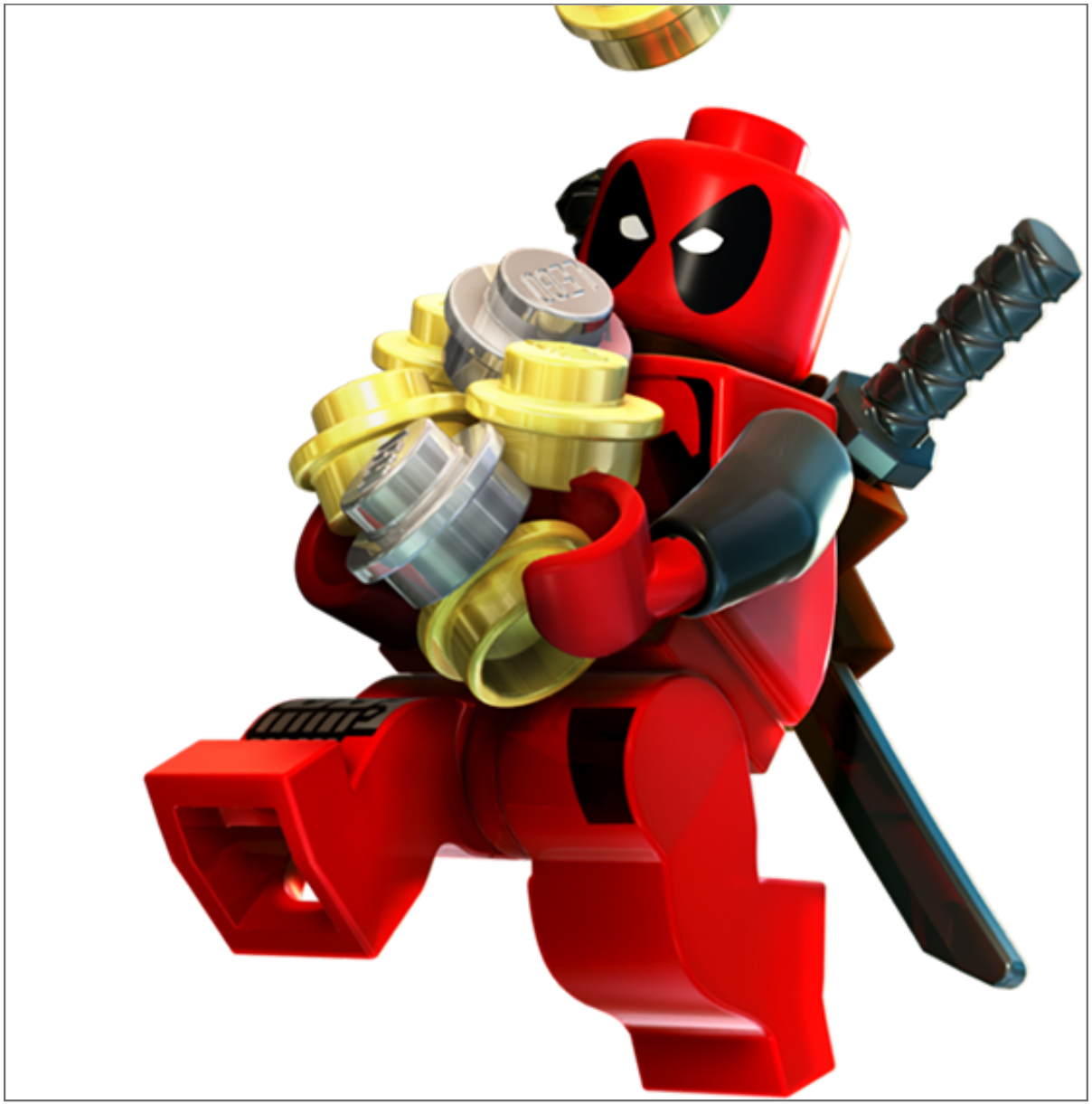


I was pleasantly surprised

What Is My Goal?

1. A harmonious development environment for myself and any development staff at my company
2. A unified ~~theory~~ practice of development, for consistent cross-platform and cross-app development behavior
3. Making all application logic and data consistently accessible across separate, but organic, systems
4. Making contracted development work more easily performed
5. Automating everything I can

I Want It All



Target Audience

You should be a Domino/XPage developer with a solid understanding of the XPages development and the Notes/Domino API.

I assume you know:

- what is, and how to implement, a Managed Bean
- what can be a data source in an XPage
- what a CSS framework is (what Bootstrap3 is/does)
- what a client-side JavaScript framework is (**AngularJS**, **Ember**, **Backbone**, **Dojo**, etc)

Topics

Segregation of Application Into Service Layers

- a decoupling of the layers of your application into micro-services
- separates the primary application logic into its own classes
- keeps the UI logic all at the UI level
- keeps styling and presentation out of the logic

This helps to keep all your code readable by not just yourself, but others.

Application Layers (as I see them)

- data store / DB (NSF)
  - provide **data** to
- primary application classes (server elements for data objects, controllers for behavior, server actions)
  - provide **application behavior** via
- servlets (RESTful or otherwise), XPage bindings (simple, EL to bean, or data object)
  - provides **interface** to
- UI layer (where the client-side logic lives)

What **M-V-C** Is

A great development architectural practice, which segregates your code into aspects, which govern solely:

- the interactions of your data objects with the data store (Model)
- the interaction of your data objects with your presentation layer (View)
- the interaction of your **application logic** (Controller, where the rubber meets the road)

Key Take Away

In XPages, out of the box, we have controls (*xp:* and *xe:*) which let us add widgets into/onto our presentation layer (the XPage or Custom Control) and script in (via *JS*, *SSJS*, or *EL*) application logic via events, partial refreshes, rendering properties, validation blocks, and other bindings.

SSJS is perfectly fine for simple applications, but can easily lead to *spaghetti code*™ on larger applicaitons with more complex logic structures.

Spaghetti Code™?

Novice thought: "But I put my (SSJS) code into script libraries."

Long story short, **SSJS is a crutch** (at least for non-trivial applications).

It works, but **the overarching answer** is to **use the Java roots** of XPages for **better performance, structure**, and sanity (your mileage may vary).

But Why M-V-C and What If I'm Not Comfortable?

My recommendation: **start small, just start**.

- create your controller class
- make it a managed bean
- use it (via EL) to perform your rendered, workflow, and validation computations

This should help you get your feet wet. With XPages almost exclusively (**OoB**) using the NSF as the data store (with potential ComputeWithForm for loose *schema*, aka- Model) and controls as the presentation(V), this means that you only *need* to use a controller for an *abbreviated M-V-C approach*.

Example

Simple validation via managed bean.

```
package com.eric.sample

import com.everything.i.need.*;

public class AwesomeController implements Serializable {
    ...
    private String someAwesomeProperty;

    public AwesomeController(){}

    public boolean validateMyProperty(){
        boolean tmpVal = false;
        if( this.someAwesomeProperty != null ){
            tmpVal = true;
        }
        return tmpVal;
    }

}
```

Example

Binding to a Control in an XPage

```
< ?xml version="1.0" encoding="UTF-8">
<xp:view>
    xmlns:xp="http://www.ibm.com/xsp/core">
    <xp:inputText
        id="inputText"
        value="#{myBean.someAwesomeProperty}"
        validator="#{myBean.validateMyProperty}">
    </xp:inputText>
</xp:view>
```

But Why Start With a Managed Bean?

Your application logic is now easily used in:

- XPages
- a RESTful servlet
- other service

Your code will be **more easily maintained, read, and ported to other systems or applications**.

Structure Your App Now

It beats the heck out of hacking on functionality while searching for where the relevent code is, without being certain you got it all.

Discussion: M-V-C / App Structure

- Keep on target with application structure
- Do you use M-V-C principles?
- What are your positive/negative experiences?
- Lessons learned
- Alternative approaches?

CSS Frameworks

The **Wikipedia article on CSS Frameworks** lists 20, and that is in no way a complete listing.

CSS Frameworks

...are pre-prepared software frameworks that are meant to allow for easier, more standards-compliant web design using the Cascading Style Sheets language. Most of these frameworks contain at least a grid.

Layouts With Benefits

Remember that **snafu with the XPage Mobile Controls** a little while back?

IBM did, after a period of time, get an Interim Fix to correct the breakage. It also was an issue unique to iOS 8, but **this sort of thing has happened before, with Android**.

Why do I prefer CSS based widgets in place of an OS specific control? Web standards mean that a (properly implemented) CSS framework won't just break on a whim. Long story short, the more web standards compliance we have can minimize any uniqueness of implementation (and potential for random breakage).

Aka- my app shouldn't break, just because my user upgraded their OS.

Popularity

Bootstrap (formerly Twitter Bootstrap) has gained considerable popularity in the web application space.

Currently the most starred **project on GitHub**.

It has also gained considerable adoption in the XPage community, spurning many to make use of custom (manual) adaptations and **Bootstrap4XPages**.

The November 2014 release (901v00_10) of the **Extension Library includes Bootstrap** and jQuery.

Other Than Bootstrap

There are other CSS frameworks than Bootstrap, each with their advntages or dependencies.

One great example is **ZURB Foundation**. Many regard Foundation as an excellent framework, but one with which to build your own version of a CSS framework.

Both Foundation and Bootstrap have class definitions with required tag wrappings and are customizable. A different framework can fit your needs, but Bootstrap definitely is the popular kid (and posisbly the easiest to implement in XPages, at the moment).

There are more CSS frameworks than I can count or name here.

CSS Grids

Grids are designed to make the responsive nature of CSS frameworks behave in a consistent fashion, across devices and screen sizes (and resolutions), according to their tags assigned classes.

The responsive grid behavior, as **demoed by Bootstrap**, illustrates this somewhat clearly. My recommendation: open the link in a desktop browser, and scale the size of your browser window to view how it adapts to different resolutions.

Bootstrap Specifics

By default, requires jQuery.

Unless you use another JavaScript library/framework adaptation, like **UI-Bootstrap**, an AngularJS based implementation.

You *can* use the Bootstrap CSS without jQuery (or another JS implementation, like UI-Bootstrap), but it will break any JS dependent features.

Comes with some nifty icons, glyphicons. These are nice, but certainly far from all-inclusive.

Font Icons

A huge door opens up when the need to download (most) images for styling goes away. By having vector based icons as a font, implementation is as easy as assigning a class to an HTML tag, and away you go. This is done all without the dependency on any JavaScript or additional XHRs to load images (like many of the Dojo widgets rely on for states, like hover, such as dijit.form.Button, bane of my existence).

My favorite icon font is **Font Awesome**. It "was designed for Bootstrap" and has lots of goodies.

Which only requires:

```
<i class="fa fa-beer"></i>
```

Discussion: CSS Frameworks / Bootstrap

- Keep on target with CSS frameworks
- Do you use a CSS framework?
- What are your positive/negative experiences?
- Lessons learned
- Alternative approaches?

A JavaScript Framework Explosion

There are many JavaScript frameworks and libraries that can make our lives as developers easier, and make the applications we build more:

- modern
- mobile friendly
- capable of using things like **Web Sockets** ("real"-time data!)
- and other fancy new-fangled things which automate what used to be more difficult
- plus more than I can fit here

But Why?

Many of these frameworks take an MVC/MVVM/MV* approach. For the same reasons you should structure your code into either MVC or MVC-like structures are the same reason you should do so for your client-side code.

It's not just for compartmentalization, it's for sanity.

Quora discussion on the why and confusion of picking a framework/library.

Sample Features: From AngularJS

What makes AngularJS awesome (and there is definite commonality with most frameworks) generally boils down to a few key things.

- bi-directional data binding (all data models by default auto-update their other references on data change, within the scope)
- templates (via ng-include or ng-route)
- OoB directives, services, filters, and more
- dependency injection
- unit testing (AngularJS was developed with e2e testing in mind, and docs examples include **protractor scripts**)

Bi-Directional Data Binding

A key concept of AngualarJS is to automate data updates (e.g.- no calling partialRefresh events, the data *lives in the DOM*, it it automates the update)

Result Edit in JSFiddle

HTML Edit in JSFiddle

```
<div ng-app>
    <input type="text" ng-model="name" /><br />
    Hello {{name}}!
</div>
```

Templating (client-side dynamic content)

Result Edit in JSFiddle

HTML Edit in JSFiddle

```
<div ng-app>
    <div ng-include src="'someStuff'"></div>

    <script type="text/ng-template" id="someStuff">
        <h5>Hello from a dynamically injected HTML
template!</h5>
        These are used in dynamic content locations
, such as routing for application state.
    </script>
</div>
```

Filtering (simple conversions without DIY)

You can always make custom ones (same as directives.)

Result Edit in JSFiddle

HTML Edit in JSFiddle

```
<div ng-app ng-controller="myCtrl">
    <span>{{ dt | date: 'shortTime' }}</span>

    <script type="text/javascript">
        function myCtrl($scope) {
            $scope.dt = new Date();
        }
    </script>
</div>
```

This was the first JavaScript I *wrote*, just to set a *Date()* object into **scope**.

Just a Taste

I could go on forever with simple examples. The bottom line is:

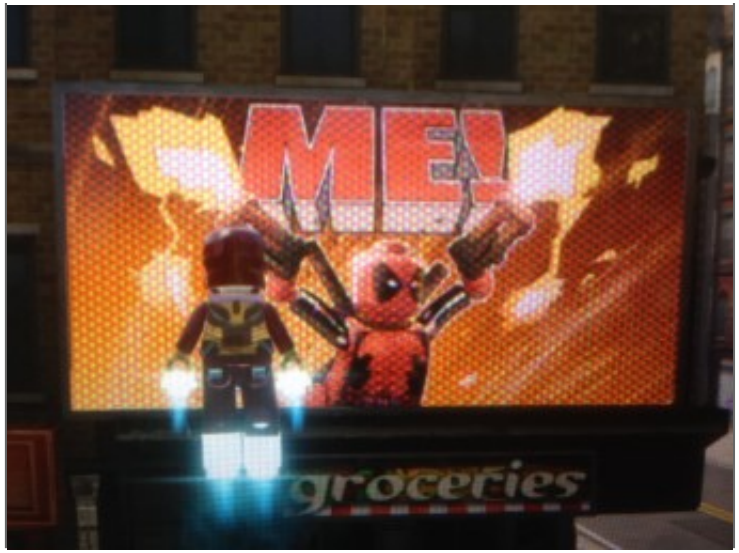
If you can automate as much as possible with a framework that fits your needs, do it!

Why reinvent the wheel? We're developers, we automate code all the time. It's just that this time, we have a lot of handy tools already available for us.

Discussion: JS Frameworks

- Keep on target with JS frameworks
- Do you use a JS framework (or library)?
- What are your positive/negative experiences?
- Lessons learned
- What made any one work best for you?

Links About Me



- **@edmoose**
- **my Dev Blog**
- **about.me**

General Links

- **Stack Overflow**
- **OpenNTF**
- **XSnippets**
- Many, many, many blogs by developers across the spectrum

Java/MVC Links

- **Jesse Gallagher**'s **Building an App with the Frostillic.us Framework**
- **Frostillic.us Framework on GitHub**
- **Pipalia MVC blog posts**
- all managed bean information (it forces a simplified segregation between the XPage and logic of your code)
- this may be the **best kept secret** in XPages development (we know it's a good practice, we just need to do it!)

Bootstrap Links

- **Bootstrap4XPages**
- **Daniel Friedrich's Blog with a more custom approach**
- **Notes in 9 136: Using XPages with Bootstrap 3 and FontAwesome**
- **Bootsnipp**
- **XControls** Teamstudio UnPlugged compatible XPage controls based on **BootCards**
- Many, many more

AngularJS Links

- **Marky Roden's AngularJS blog series**
- **Mark Leusink's blog**
- **Sven Hasselbach's blog**
- **AngularJS PhoneCat tutorial**
- **Shaping UP with AngularJS** (a free CodeSchool course, sponsored by Google)
- **Egghead.io** (site with free and subscription videos on AngularJS and other tools)

Bring It Home

Closing Discussion

Questions?