# Introduction to MVVM & Knockout.JS

## Model-View-View Model (MVVM)

* Design pattern developed by Microsoft for use with the powerful binding engine of Silverlight and WPF
* Similar to MVC pattern (Rails, ASP.NET MVC, …)
* Separation of functional development provided by MVC as well as leveraging the advantages of data bindings and the framework by binding data as close to the pure application model as possible
* **Model**: domain model (object oriented) or data access layer (data-centric)
* **View**: presentation layer (GUI)
* **ViewModel**: conceptually similar to the Controller in MVC but positioned closer to the UI for more immediate execution via framework bindings
  + Controller = set up data and merge into view; receive data from view and process
  + ViewModel = continually interacting with—and usually updating—the UI.
* Declarative Binding: Associate elements on the view with data and/or functions in the view model.

## Knockout.js

* JavaScript library for application of MVVM pattern
  + Declarative bindings
  + Automatic UI Refresh
  + Implicit Dependency Tracking
  + UI Templating
* Free, Open Source (MIT)
* Steve Sanderson – Program Manager on ASP.NET team at MSFT
* Requirements: HTML & JavaScript

## Knockout Bindings

* Declarative
* Iterative
* Eventing
* Templating
* Observables
  + ko.observable(value)
  + ko.observableArray([…])
  + ko.computed(function() { … })
  + ko.applyBindings(ViewModel)

## jQuery

* John Resig – Khan Academy
* Free, Open Source (MIT/GPL)
* JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development
* “Write less, do more”
* Plugin model w/ tons of community content (some of it’s even good…)

## Additional Resources

KnockoutJS Official: <http://knockoutjs.com>

Community support site: <http://knockmeout.net>

Steve Sanderson Demo: <http://channel9.msdn.com/Events/MIX/MIX11/FRM08>

Brad Wilson Demo: <http://bradwilson.typepad.com/blog/2012/07/webstack-of-love.html>

# Demo Notes

## Part 1: Simple editor

1. Create View Model and enable data binding



1. Create view elements

It is important to note that some elements are bound to different attributes. The input element is bound to the ‘value’ attribute whereas the span element is bound to a ‘text’ attribute. This is subtle but important distinction. You can think of the text property as innerHtml in JavaScript; all input elements that support innerHtml seem to enable binding through the ‘text’ attribute.



\*Notice that data bindings are one directional; changes to the textbox values do not affect the text displayed in the span

1. Add observables to view model

The view is now engaged in a 2-way binding with the view model



1. Add observable array to view model



1. Add view element to display array



## Part 2: Adding Functionality

1. Create an “Add” function to the view model to push a new person entry into the people array



1. Create button element and bind to “Add” function



1. Enable items to be removed from the list





\*Notice the $root keyword on the data binding of the ‘a’ tag. This is necessary because the list is being bound to the people array, not the root-level view model. The $root keyword enables us to pull from the root-level context. Another option which would work is the $parent keyword; which gives you access to the direct parent of the current context.

1. Use computed values to display full name





1. Disable button element based on state



The valueUpdate binding tells Knockout to send binding notifications after the user presses a key inside that element. This is as opposed to the default behavior where notifications are pushed on loss of focus.

1. Add drop down list

Add an array to the view model to store the option values. Add another observable property to the view model to store the selected value of the drop down. Also, add a property to the Person constructor to enable the association of the selected value with a person and modify the ‘name’ computed property to display the new value.



Finally, create the view element and set up the binding properties.



1. Declarative CSS

