**Introduction**

*This is the introduction and uses the first 4 slides. There isn’t any scripted content for this section, as you just expand on the slides.*

[Slides 1,2,3,4 -> Intro material ]

**Comparison of SPA Vs. Traditional Web Applications**

*This is the framework introduction and uses slides 5 through 17. You should be familiar with the text below which is the information to talk about over these slides. Slide position is mentioned where possible.*

[Slides 5,6,7 -> Variance in composing pages / page layout ]

There have been various methods over the years for composing pages. There are very traditional includes, such as in PHP, ASP, etc.

In the J2EE world there has been JSTL for includes, and libraries like Tiles for composing pages.

Even before all of this, designers and developers were using frames to try and avoid rebuilding the entire page.

With an SPA, even when a template engine isn’t used, the entire page is rendered once, and from there only the portions of the page concerned with what is changing are updated. Users spend an increasing amount of time interacting with an application before it has to go back to the server. The blink of the page render is gone and replaced by progress bars or animations.

Responsibility also changes. The server can now focus on areas like services and security. Server side developers can focus on data storage and service interfaces without knowing anything about the look, feel or even design of the front end. Services can be designed to be a mix of public and private APIs.

On the front end, the responsibility for page composition is taken on, as well as routing logic and some business logic. It should be noted here that what doesn’t change, is the need to validate all data arriving at the server. Do not calculate the shopping cart total on the front end and then trust the value ☺.

For developers, there is much more clarity of role. Work can be more easily divided into roles; with developers work on the server, look & feel, and front end application logic.

**Framework Tour and Comparison**

*Compare the frameworks based on their usage and features, then give a deeper overview of each framework.*

[Slides 8,9,10,11 -> Basics framework comparison]

There are a growing number of frameworks for developing client based applications, whether “SPA” style or not. We’re going to look at a couple of them and talk about their popularity, differences and finish with some warnings about relying too heavily on these frameworks.

All of these frameworks work to provide a structured MVC/MV\* pattern for building applications. Each usually provides more or less features with the trade being control of implementation and architecture or ease of use in the basic case.

Market Share **@TODO**

Trends and other Info **@TODO**

**Angular.JS** [Slide 12]

Angular is a library currently supported by Google. It’s widely popular and based on adding additional markup to your html. By placing markup you give Angular instructions and information about your page.

Angular is one of the larger frameworks available, because of all the features it provides. It’s Very declarative and inspired by ideas like Silverlight, which is exemplified by the two way data binding and html markup control. Angular has no dependencies, but wants to stand on its own; so it doesn’t work well with other technologies like JQuery.

= Still more to add here =

**Backbone.JS** [Slide 13]

Backbone is a small library that provides just enough structure to assist you in building small to medium sized applications. As application sizes grow, Backbone requires more and more architecture on your part.

Being one of the smallest libraries available, you get the benefit of fast review of the source code and easy handling of special cases. Backbone provides Routers as a primary controller, and is considered a MV\* library.

Backbone depends on Underscore, and you will want to provide a templating library; usually Handlebars, Mustache or \_.template.

**Ember.JS** [Slide 14]

@TODO

**Knockout.JS** [Slide 15]

@TODO

**Data Binding and Conclusion** [Slide 16, 17]

One of the differentiating factors with framework and libraries is the amount of binding they offer. Developers are split on this idea. Some think the inclusion of binding is a necessity, and others feel that binding depends on the project needs.

One-way : Either changing the model updates the view, or changing the view updates the model.

Two-way: Changing either the model or the view, updates the other.

We’re going to be working with Backbone. It’s the easiest framework to plug into an existing application, and even when you want to use a larger framework, the ideas learned from Backbone are a good starting to organizing front end code.

**Closing (30 minutes)**

We’ll take 30 minutes to try and answer form frequently asked questions about SPAs. Hopefully this brief talk will prime attendees for asking questions and talking about the exercises and how they solved them.

FAQ entries:

1. Different front ends for mobile vs. desktop.
2. Deploying SPA’s as part of a class deployment process.
3. Development environment.

Depending on questions and talking by attendees, there will be a summary. We’ll show a list of additional resources on the board and probably as part of the *handout?*

**Sessions**

This portion of the document pertains the exercise slides.

**Narrative:**

Your boss wants to quickly deploy a restaurant reservation system to take advantage of a new public rest api created by the chamber of commerce. It allows anyone to get information about restaurants and make reservations. The chamber of commerce plans to expand this service in the future, so he wants to get in on the ground floor.

Since your boss wants this application quickly, so he has already had the web designer draw up a mockup for you to use in building the application.

Since you’ve been hearing about building SPAs and learning about Backbone; you’ve decided to use Backbone to build the application. Most of the information here is ‘points to remember as the presenter’.

**Session 1 :**

* Tell everyone that they can begin typing and you will come back.
* Also let them know that we have tags for each solution to each step.
* Talk about the directory structure.
* Show result of each exercise first.
* Will be working through:
  + Scope/Closure module.
  + Explain \_.extend, which is kind of the basis for Backbone.
  + Basic Model.
  + Basic Collection.
  + How this is wired to REST.
  + Attendees will use the try\_it.html for this exercise!
  + They will not write a main.
* Explain that the fetch is asynchronous.
* Handlebars: bring up the value of client side templating. And not having to dom manipulate via jquery

**Session 2 :**

* Will be working through:
  + Small introduction to Handlebars
  + Restaurant View
  + Restaurant List View
  + A simple main in application.js
    - Will need to talk about the various ways to do this (router, simple main, routed tabs, etc).
  + Attendees will switch to index.html and the themed application!
* Might mention using Marionette list view.

**Session 3 :**

* Will be working through:
  + Router and Restaurant route.
    - Be sure to talk about the ready function at the bottom that isn’t part of the router.
    - You might additionally mention Marionette here.
  + Add the ajax call to fetchRestaurants.
    - **We are also waiting until restaurants are loaded to render. (See below!)\***
    - **We are showing that you don’t have to use Backbone’s built in REST!**
  + Add the simple TimeSlot View.
    - We’re showing use #each instead of a ListView
    - The template is shown in a later slide so it can be directly compared!
  + Update RestaurantView to show TimeSlots.
  + Introduce Events!
    - Attendee will add code for selecting a time, but we’re just console.log’ing this so that we can talk about events and split off some of that work. **The focus should be on the Event Vs. Route difference.**
  + \*Update the RestaurantListView to wait until everything is done loading and send out notification. **The ready trigger is important because it allows for bookmarking the restaurant selection. The app will wait to start routing until the Restaurants have been loaded.**
  + Will now be at the comparison slides, showing the Handlebars template for the Timeslots so you can compare using #each with using the ListView

\**When creating the router, you may also have to modify the handlebars template to use your route. This change isn’t shown, but we can help you with it if you are stuck*.

**Session 4:**

* Will be working through:
  + Create the simple Reservation Model (can move quickly here!)
  + Create the simple Reservation View.
    - Key point: giving it a restaurant/time.
    - It’s a FORM!
    - We’ll add events/binding/validation later! (if asked)
  + Modify the restaurant view to show the reservation form.
    - Many ways to do this.
  + Back button discussion
    - Waiting for things to load allows for bookmarks.
    - Using routes for a controller allows for back/forward.
    - Using Events allows for direct control or when you don’t want back/forward.

**Session 5:**

* Will be working through:
  + Adding Events:{} and corresponding methods to Reservation.FormView
    - Be sure to talk about the JQuery selectors.
    - Be sure to mention binding frameworks.
    - Be sure to mention Binding VS Gathering.
  + Create the Reservation view.
    - Fetches everything.
    - We are fetching so that this can be a route that can be bookmarked/etc.
  + Add the Reservation view route to the Router
    - Could be done with a top level view that get rendered, but we left the DOM manipulation for Brevity. You could also use Marionette.

**Session 6 :**

* Will be working through:
  + Will use the backbone.validation plugin.
  + Will talk about extending the backbone.validation callbacks.
  + Talk through the CSS/HTML for the error div and css classes.
  + Add Validation:{} to the reservation model.
  + Update the Reservation.FormView to bind to the validation and then do validation before submitting the form.

**Session 7 :**

* Will be working through:
  + Introduce QUnit.
  + Giving them a first test and a bunch of empty tests.
  + They can fill in the empty tests, etc.
  + Will talk about testing views, but not actually demonstrate unless time.

You should be returning to power point…. (Scroll back up to power point section)