Demo 1: jQuery, jQuery UI, Amplify

1. File->New Project
2. Templates ->Other Languages->JavaScript->Windows Store
3. For those of you who are totally new to Windows Store development, when we spin up a new project we get:
   1. Default.html
   2. Default.js
   3. And Default.css…and I am just going fill this bad boy up while we are in here **[c11]**
4. First thing we are going to do is use NuGet to bring in jQuery
   1. **“install-package jquery”**
   2. When that completes you’ll see we now have a “Scripts” directory and it contains all the jQuery files
   3. And since I am in here I’ll pull down jQuery UI just in case we want to use that in our demo
   4. **“install-package jquery.ui.combined”**
   5. So now we have jQuery UI as well.
   6. I am also going to use one of my other favorite libraries, called AmplifyJS, that has a lot of cool functionality, the specific piece we are going to use in this demo is the ability to easily create reusable AJAX calls.
   7. **“install-package amplify”**
5. Next I am going to create my ajax definitions file for amplify to use.
   1. So in the /js directory I will create a new JavaScript file named **“ajax-definitions.js”**
   2. **[j11]**
   3. And here I am putting
      1. The base url for the services I am using, and for this demo we’ll be using the GitHub api.
      2. And then I have to amplify requests defined
         1. The first one for getting the user
         2. And the second for getting the repos
      3. We’ll see how this works in a bit.
6. Now I am going to create another script file that will hold the meat of the application.
   1. Again in **/js** I’ll create a cleverly named file called **script.**
   2. **[j12]** And then I’ll create my document ready function and a static array of GitHub user id’s.
   3. Next, outside of the document ready function I will create a function to pull the data from GitHub **[j13]**
7. Alright, now we can jump over to our Markup and…
   1. Pull in our references
      1. jQuery
      2. jQuery UI
      3. Amplify
      4. Ajax-definitions
      5. Script.js
      6. And Content/Themes/base/jQuery-ui.css
   2. And create our markup **[h11]**
      1. Discuss the code
8. Run it…and see that it works….snazzy!
9. Let’s add some more!
10. When one of the users is clicked I want to display another view with two tabs, one for the profile and one for the repos. I already have the markup so let’s go back in to **script.js**
    1. **[j14]** start by creating a couple of functions outside the doc ready and these two functions load the profile and repos tab.
    2. Now back up in the document ready function…
    3. **[j15]** we’ll setup a click on the images displayed on our initial screen.

Demo 2: Knockout

1. File->New Project
2. Templates ->Other Languages->JavaScript->Windows Store
3. First thing we are going to do is use NuGet to bring in jQuery
   1. **“install-package jquery”**
   2. And a little KnockoutJS
   3. **“install-package knockoutjs”**
   4. And finally requireJS for module loading
   5. **“install-package require”**
4. And now I will open up “default.html” and pull in **requireJS**:
   1. Show of hands, how many of you have used requireJS or another JavaScript module loader?
   2. For requireJS we have to set our “Data-Main” attribute which is the main entry point of our application
      1. For us it will be a script file that will reside in the **“/js”** directory named **”app.js”**
5. Now we’ll create that file in the **“/js”** directory and…
   1. Inside it we’ll configure require. **[Snippet]**
      1. I am configuring require for jQuery, Knockout, and a custom module named “cars” that will live in “/js/modules”.
6. We’ll create that folder and file now.
7. And inside this module we need to define what other modules we’ll be using. **[Snippet]**
   1. And we’ll be using jQuery, and Knockout and we’ve all setup the associated aliases.
   2. Alright, we can save that and jump back to “app.js”
8. **[Snippet]** Where we can code up our main require method.
   1. Where we will define the 3 modules we’ll be using, jQuery, Knockout, Cars.
   2. Now, as those of you have used RequireJS in the past it can get a little crabby
   3. So I really don’t like to get to deep w/out confirming that it is happy.
   4. So I am going to put a **“debugger”** in here so I can verify that everything is Happy-Happy-Happy.
   5. And now I can inspect jQuery and Knockout, and see that they are indeed being resolved
   6. Cars is still undefined because we haven’t set it up to return anything.
   7. And most importantly require is behaving today.
9. Alright let’s jump over to the cars module and get that coded up.
   * 1. **[Snippet]** Here I am defining a variable
        1. Since we are working with knockout the first will be a knockout observable array
     2. Now we need a method to return some data. **[Snippet]**
        1. This is just a method that returns an array of JSON objects that happen to be super cars.
        2. This code is demo-ware…typically you would be making an AJAX request to some endpoint here.
        3. And we are taking that array of cars and looping through it, pushing each one into our observable array.
     3. Lastly we need to return, or reveal something from this module. **[Snippet]**
        1. We’ll return the cars
        2. And the function…with a better name ;)
     4. That’s all the code for the module, so we can run this and when it hits our debugger statement we should have the cars module reference populated….and we do!
   1. Back over in app.js we need to call the getCars function on the cars module. **“cars.getCars();”**
   2. And then I need to create a new constructor function **[Snippet]**
      1. and in it you can see we are setting **this.cars = cars.cars.**
   3. Now I need my jQuery document ready function **[Snippet]**
      1. And inside there we call ko.applyBindings passing the new App we just created.
10. Now we need some markup…so over in default.html **[Snippet]**
    1. We have an H1 with our title
    2. Then we have an unordered list that is bound to the cars collection
    3. And a div that contains spans for each one of the properties we are displaying
    4. And you can see that each of the display spans has its text property bound to a corresponding car property.
11. Now that we have some markup we can run this and we should be able to see something.
    1. And there we have our list of cars being displayed.
12. That’s cool, but the real power of Knockout is in the observable collection, that auto updates the DOM based on its changes…so let’s play with that.
13. I am going to go back to cars.js and create an Add Car method. **[Snippet]**
    1. Again demo-ware just to prove the point
    2. Our add method push’s a new hard-coded car on to the collection.
    3. Next we need to expose the add method as part of the return
       1. **add: addCar \*\*\* Don’t forget the comma on the previous line**
14. We need to expose it as part of our App object in app.js as well
    1. **this.add = cars.add;**
15. And now we need to create some markup so we can trigger the button, so over in default.html
    1. We’ll add a new button and bind its click event to add
    2. **<input type=”button” value=”Add New” data-bind=”{click: add}”/>**
    3. Now we can run this and…click the button and see that the item our new car is auto-magically added to our list.