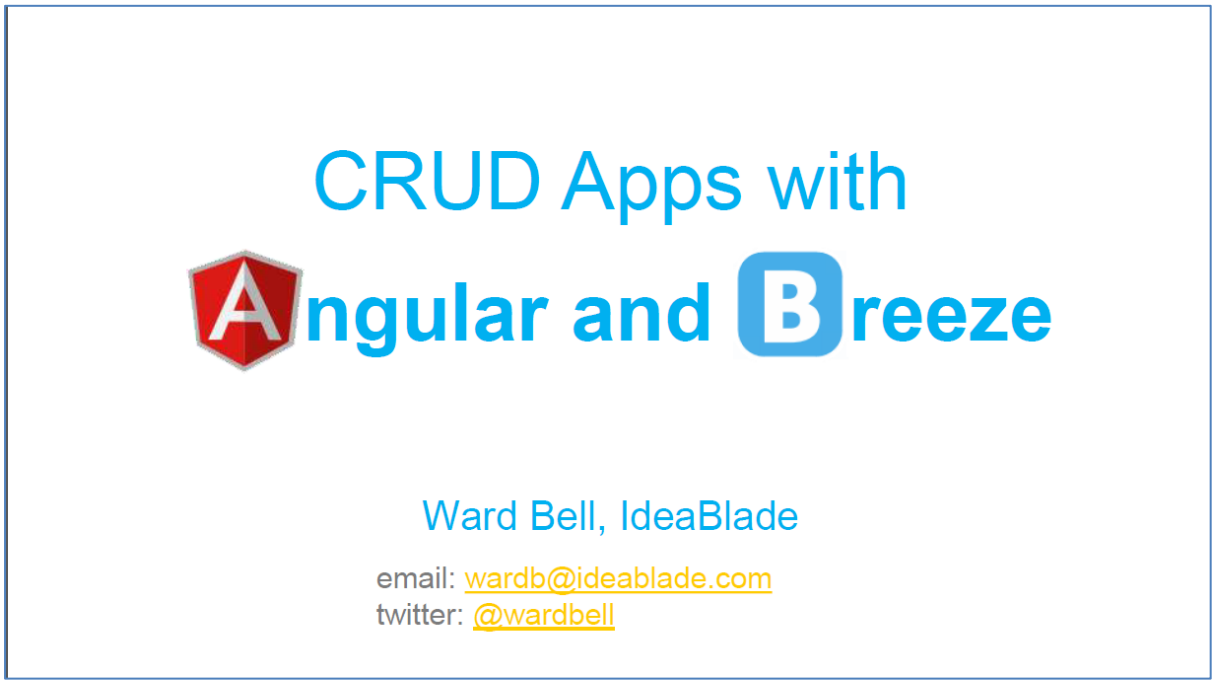
CRUD Apps with Angular and Breeze

User Group Talk Transcript

Talk given at **AngularJS MountainView** User Group on **12th March 2013**



Video: http://www.youtube.com/watch?v=P2ErSQj3SN8
Video Transcription by **Ian Smith** (http://twitter.com/fastandfluid)

***Disclaimer**: The original content is copyright of the original "free to download" video published as indicated by the links to the original source material above. Any mistakes, opinions or views in that content are those of the original presenter. Any mistakes in the actual transcription of that content are the fault of the transcriber. Please email ian.smith@fastandfluid.com if you spot any errors.*

Session Brief

$resource is a cool AJAX abstraction for retrieving data for display. But if you're building a classic Create/Read/Update/Delete (CRUD) app, you've got bigger challenges: ad hoc user queries, change tracking, client-side validation, navigating object graphs, binding to dropdown lists, caching, stashing data in local storage. That's what CRUD apps do ... and CRUD is the bread-and-butter of the working developer.

Ward Bell from **IdeaBlade** will show us how **BreezeJS** combines with **AngularJS** to meet those challenges. He promises some good code and good fun.

Contents

Introduction

I'm Ward Bell. I'm VP of Technology and a founder at IdeaBlade, a company in the Bay area.

I'm here to talk about Angular and Breeze.

Breeze is our product and Angular is a Google product and they work really well together when you're building CRUD apps, which is Create, Retrieve, Update and Delete – something like that. Anyway business apps are a very typical part of this.

I come from the Microsoft arena. We've built a lot of business apps there so building business apps is kind of what I know a lot about. I don't know how to build games but I know how to build business apps.

Because I don't know how to make games I can't be Huggy Bear, because I can't wear a lot of bling so I have to do that thing that a lot of business developers get paid for which is to build business applications.

LOB CRUD Apps

You can call them line-of-business applications: that's LOB and CRUD apps. They kind of go together because they're all about asking for data and saving data and stuff like that.

LOB CRUD Apps

ERP

CRM

Asset Tracking

Call center

In the typical category is ERPs – Enterprise Resource…. You know how you know the acronyms but can't remember the names? CRM – Customer Relations Management, Asset Tracking, call centres….

LOB CRUD Apps

Find problem orders…and fix them

Book a premium 3 week vacation

Sell or renew rail cars coming off lease

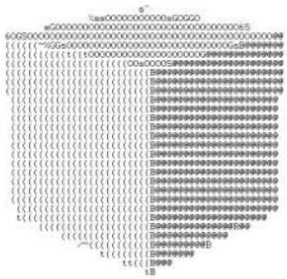
Pick sessions for a tech conference

The kinds of apps in which users sit in front of them for long periods of time and they kind of ask them questions like “I gotta go and find all the orders and fix them” or “I gotta go and book a premium three week vacation or renew a rail card” or something like that.

These are the typical tasks that people sit in front of them all day long. Really what the modality is – it’s pretty common across all of them - is that the user sits there and finds a target dataset to work on, like orders or something like that, and then the application pivots around that data set, looking at it from different angles as they process it.

LOB CRUD Apps

Find a target data set and pivot around it



They’re not surfing all over the place. They’re just fine with what the customers want and just go at it.

LOB CRUD Apps

Selected data set is small

From a computing perspective it's a small set of data, but from their perspective it's not so small because it's about what fits in their head, what fits in their attention span so they usually just grab a chunk of data and then sit with it for a long time.

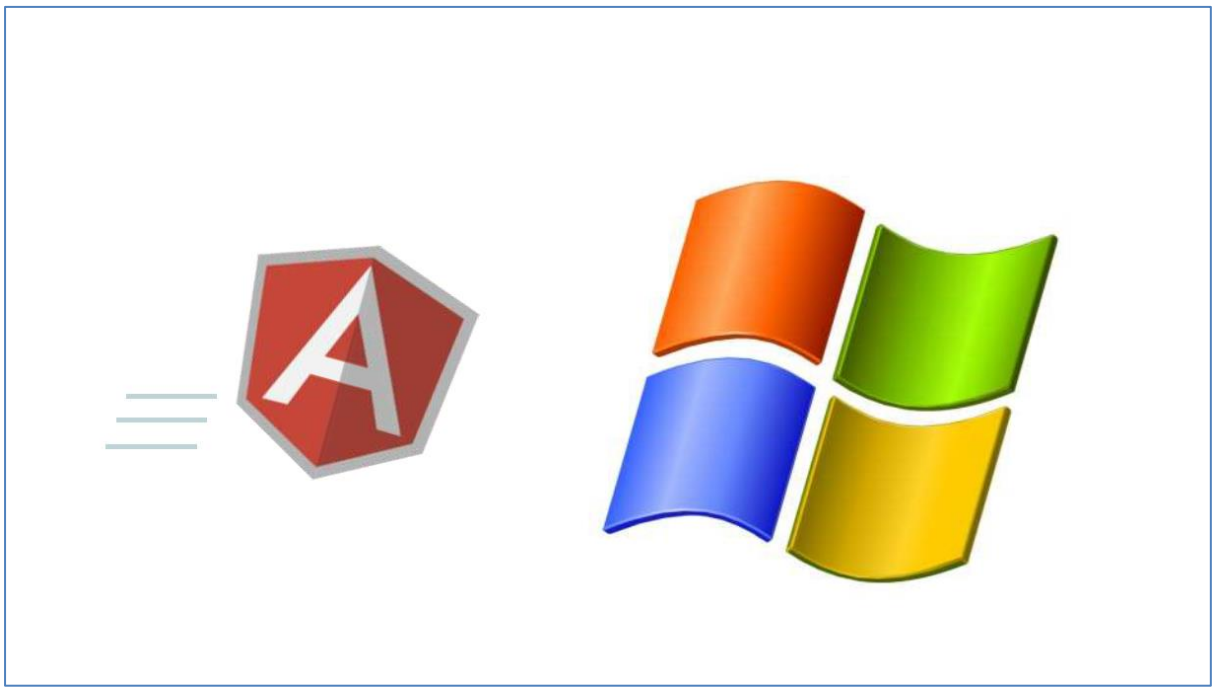
LOB CRUD Apps

Keep computing close to the most precious resource

Because they work all day long doing that kind of thing the most precious resource is the user and so the whole idea is to say "Let's move all the computing right down. Let's have client-side computing". Another term for this might be SPA – Single Page Application. You can think of it as AJAX applications, HTML/JavaScript apps – whatever you want to call it.

Client Centric App (SPA)

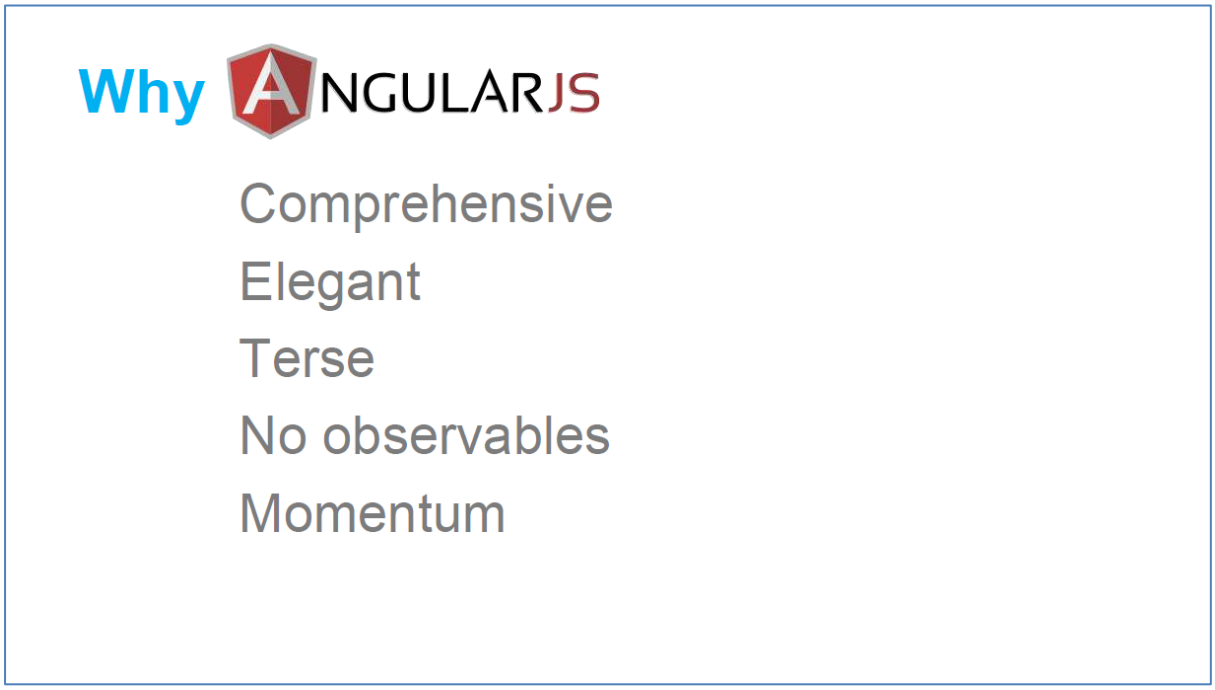
We call it SPA a lot where I come from. That's it and AngularJS ought to be really good at it because of what you just saw and it turns out it really is.



So what's happening is a lot of people who were in the Microsoft space are seeing this Angular thing coming and they're kind of interested in it.

There's a list of reasons why they are.

Why Angular?



For one it's comprehensive. So it does a whole lot of things you don't have to buy or get a whole lot of libraries. It does screen management and dependency injection and a bunch of other things that you can learn about. It's really elegant to work with which is nice because there are some other comprehensive ones that just seem like you write and write and write and write and you can't figure what the heck it was about.

It's nice and terse. Some people like that it isn't sitting there listening for observables so I don't have to have special purpose objects that raise notification events.

It has a lot of momentum. There's a lot of people working to make Angular better all the time and it's getting a lot of traction.

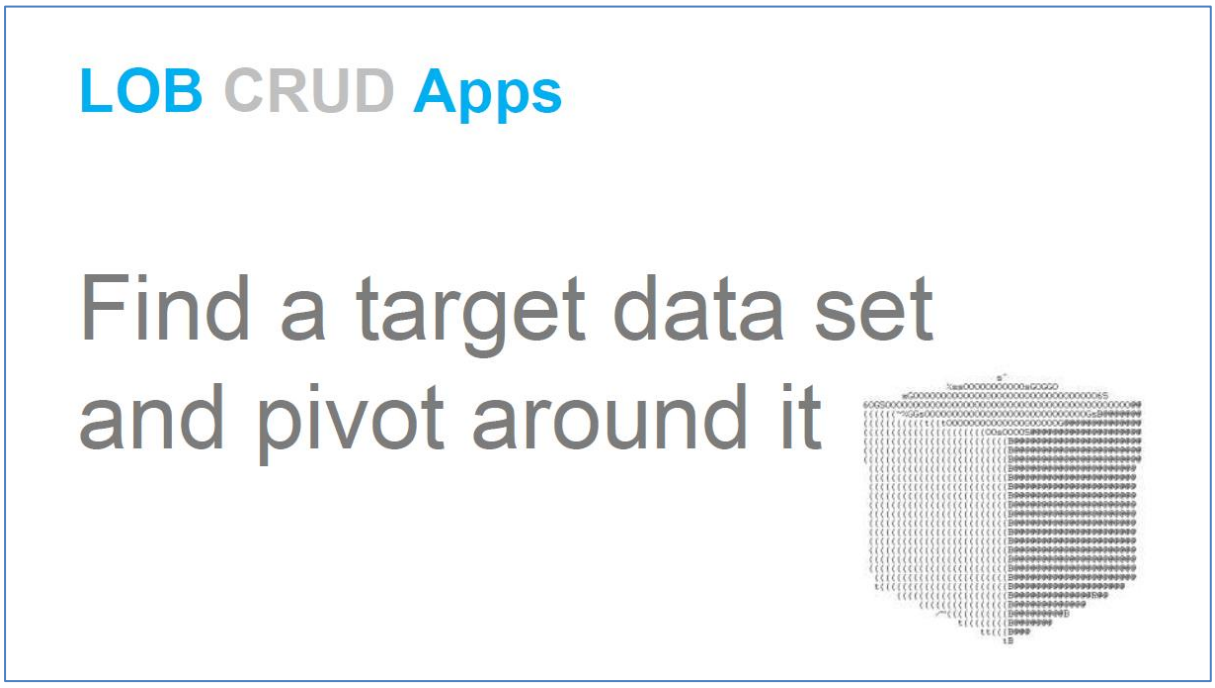
Lots of reasons, and there are people in the Microsoft place who are very well-known there who are writing about it, like my friend Scott Allen here.

Anyway you start to do it and you're the captain of your Angular luxury cruise liner and really your job is to make sure everybody has a good time and you don't end up on the rocks. That's really your only responsibility: just keep it moving. Keep them happy.

As we're doing it really if we think about that liner there's the entertainment section up above which is where you do the presentation – and that's where Angular's really super strong. And then, if you're doing line of business, there's all that data access stuff and that's where Breeze is going to come in.

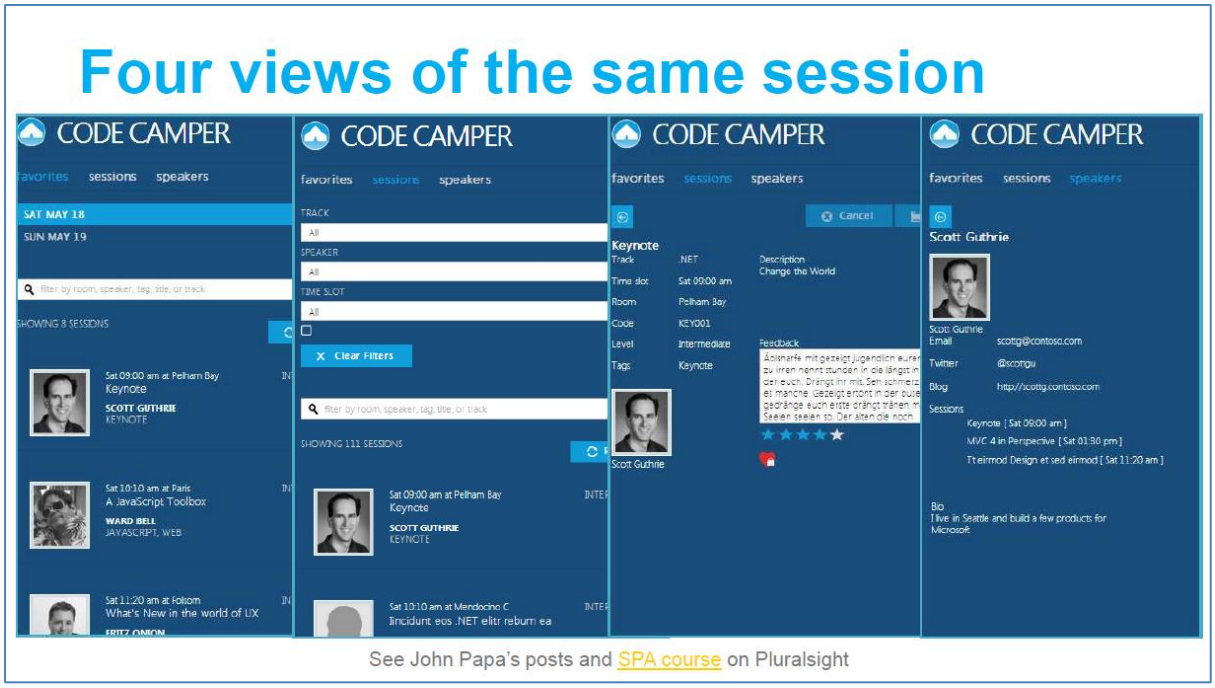
Find a target Data Set and Pivot Around It

You could say we're going to help you build applications from A to B except that doesn't sound right. Anyway, remember it's about "find a target dataset and pivot around it", so it's a lot about data.



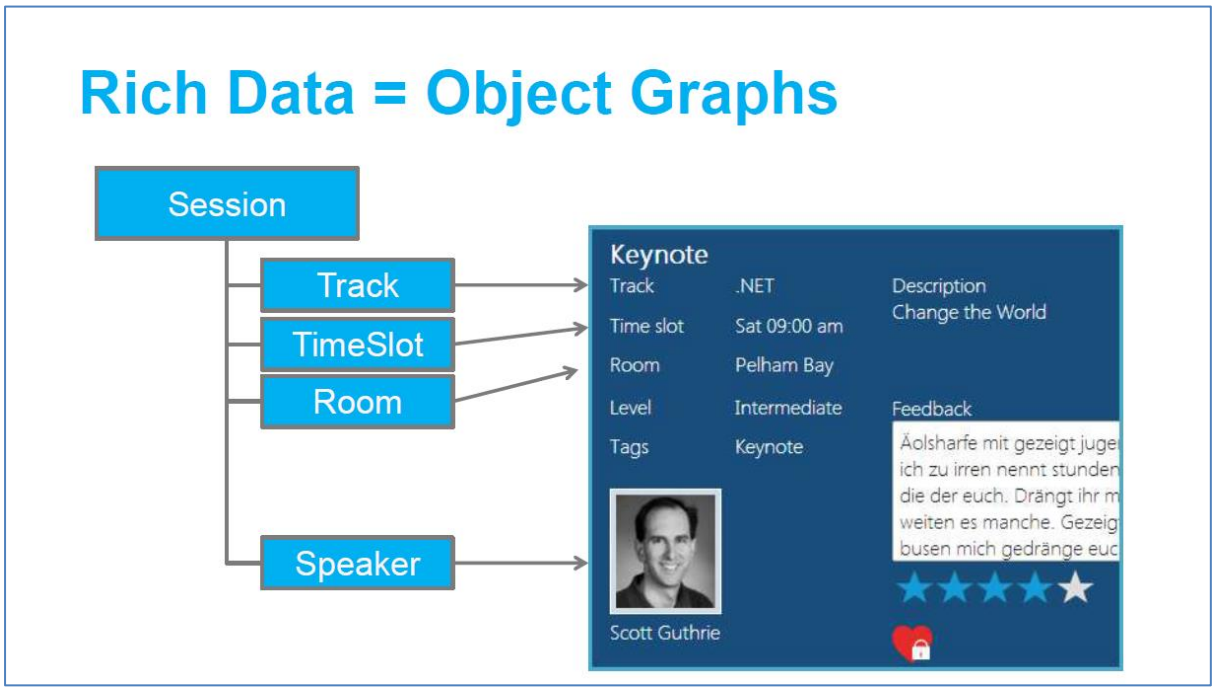
What do I mean by pivoting around it?

Suppose I had an application that was for scheduling a code camp and there were sessions. The chances are that if you were picking a session, or editing a session or doing something like that "session" keeps occurring from screen to screen to screen.



Rich Data = Object Graphs

You're really talking about the same session as you look at it from different angles and make different changes. So here we have this class by ScottGu carrying four different screens.

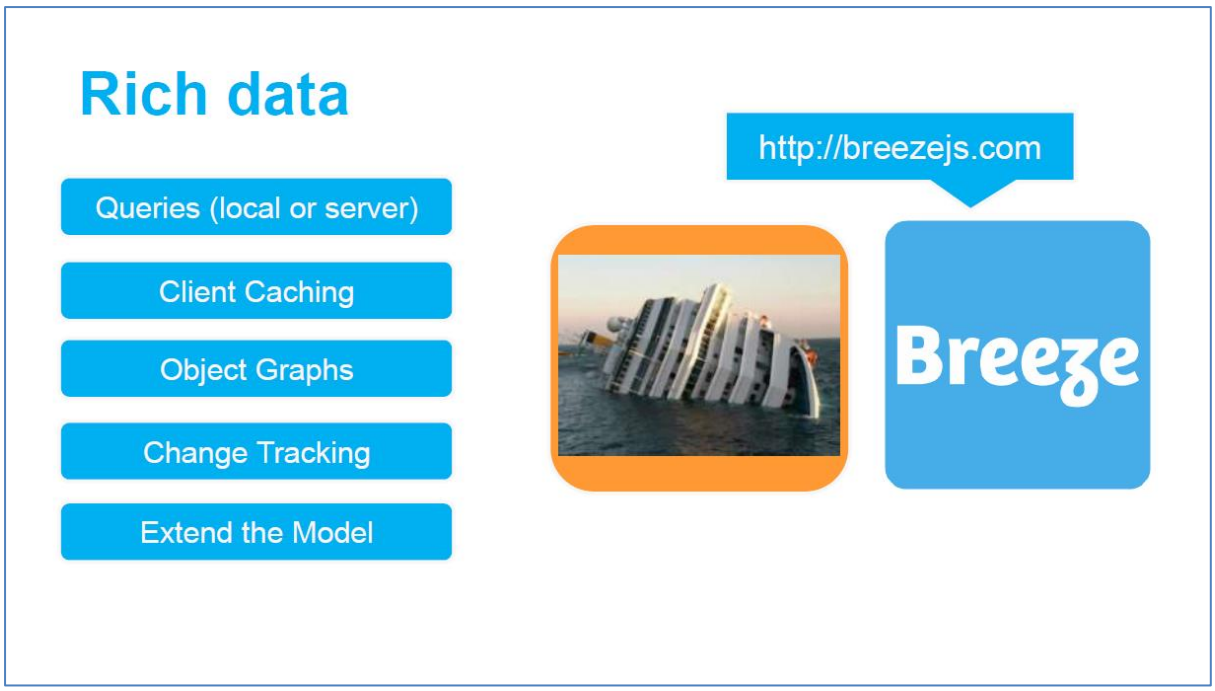


When you finally look at one of the items in it you find that the data that are behind it – it's not just one object. It's usually backed by a SQL relational database or something like that. So it has relationships to other things. Sessions have tracks that they're on and timeslots and rooms and speakers and you have to have this sort of network of related objects.

That's part of the LOB space so how are you going to deal with that if you're going to do it on the JavaScript side?

I'm going to propose to you that you're going to need to do queries, and some of them because you're going to be on mobile devices, and some of them because you're going to keep looking at them from one view to another, you'll need those queries to be both local and on the server. You'll want to be able to cache data. You'll want to be able to use those object graphs I was describing. You'll want to be able to do change tracking because they're going to be making changes and you're going to hold them – you're not going to just save them right away.You're going to have a "Save" button.

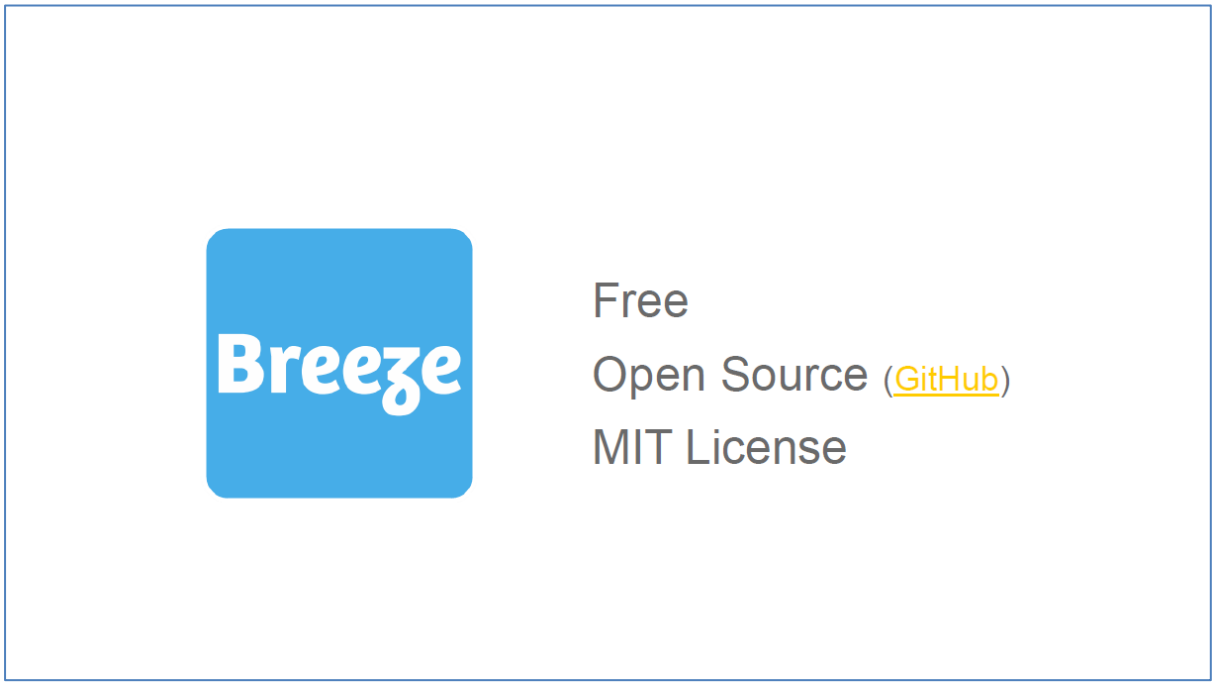
And you're going to want to make those things that you're dealing with like Sessions and stuff, you want them to have behaviour – they're not just data bags. Read them in out of the database – they're just data bags. But no, if you're going to use them they're going to have behaviour.



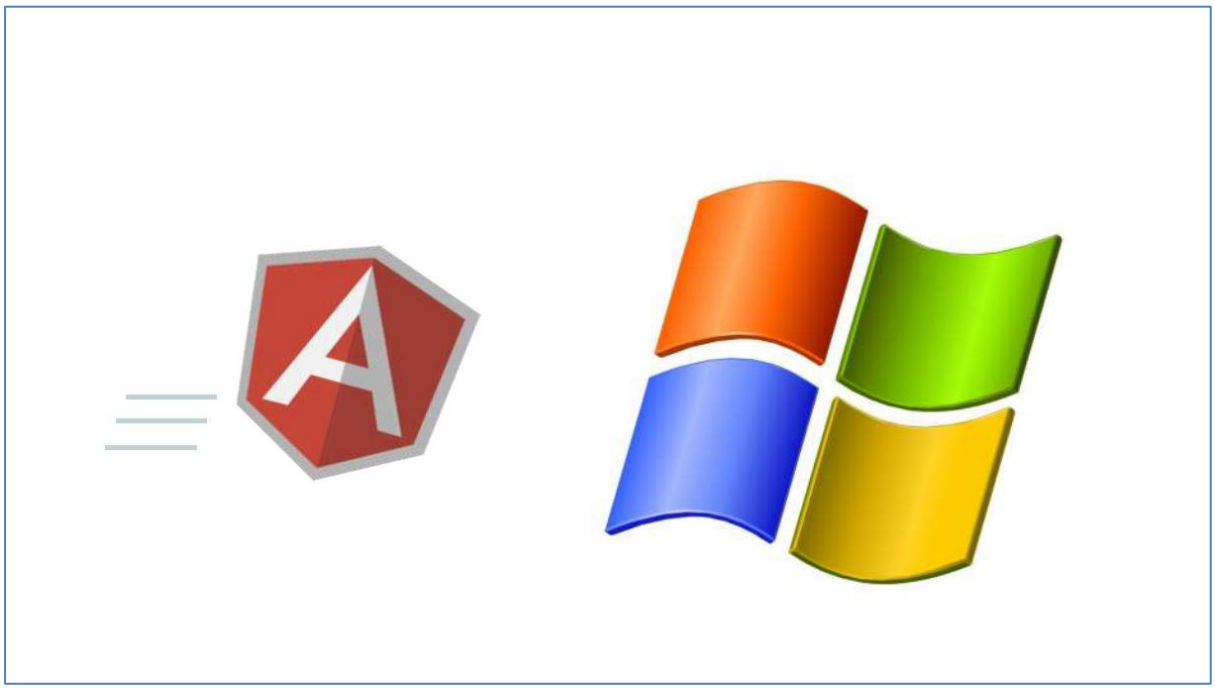
So you can roll your own framework around this. This is really what ends up happening, whether you call it an official framework or not, and then your ship can run aground.

Or you can use ***Breeze***, and that's my commercial.

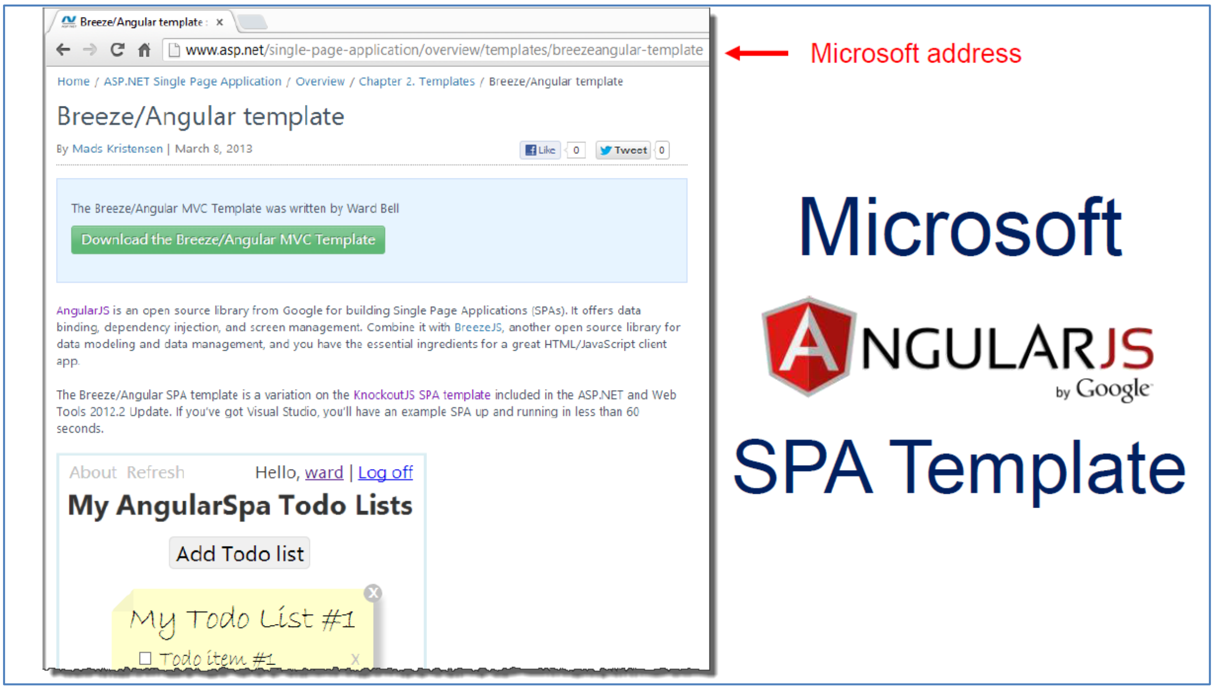
My Commercial: Use Breeze!



It is open source. It's an MIT license. It's on GitHub. Enjoy it.



So back to this Angular thing, and Angular and Microsoft. Google and Microsoft. Cats and dogs living together – what's that? But yes, it does happen.



Not only that but there is now an "in the box" .. Visual Studio is the .NET IDE of choice and right in the box now you can get a Breeze/Angular template that shows Angular working with Breeze inside of the IDE, Visual Studio.

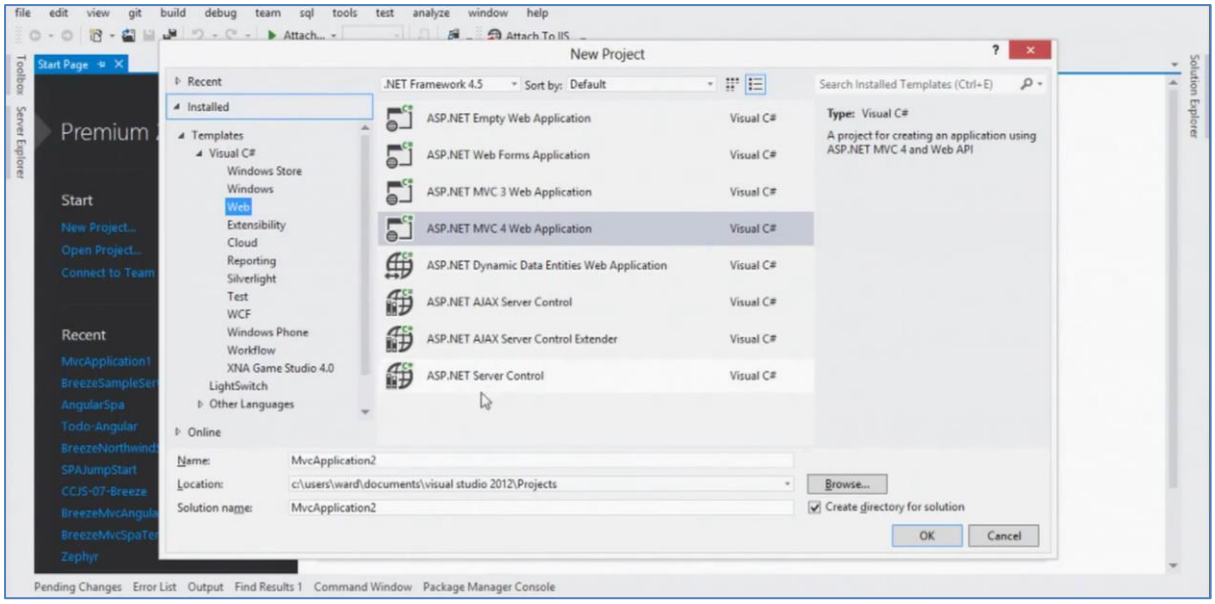
As you can see they're talking about it on a Microsoft address – not on my site. Of course I talk about it too.

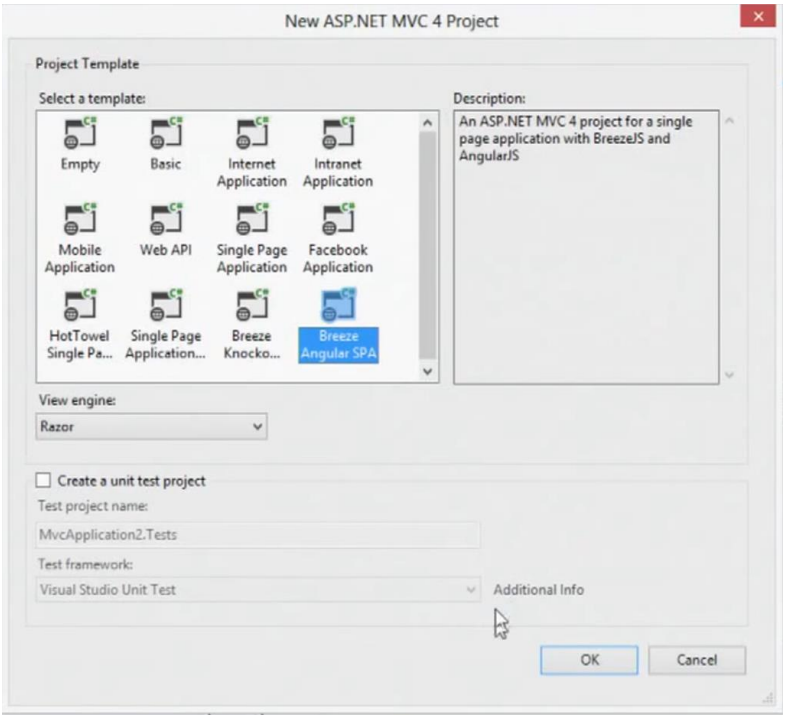
It's a two page app. It's really simple but it illustrates a lot of the capabilities both of Angular and Breeze. This is what it looks like when it runs and we're going to do a demo of it.

Then at the end I'll show you more of what you can learn about it.

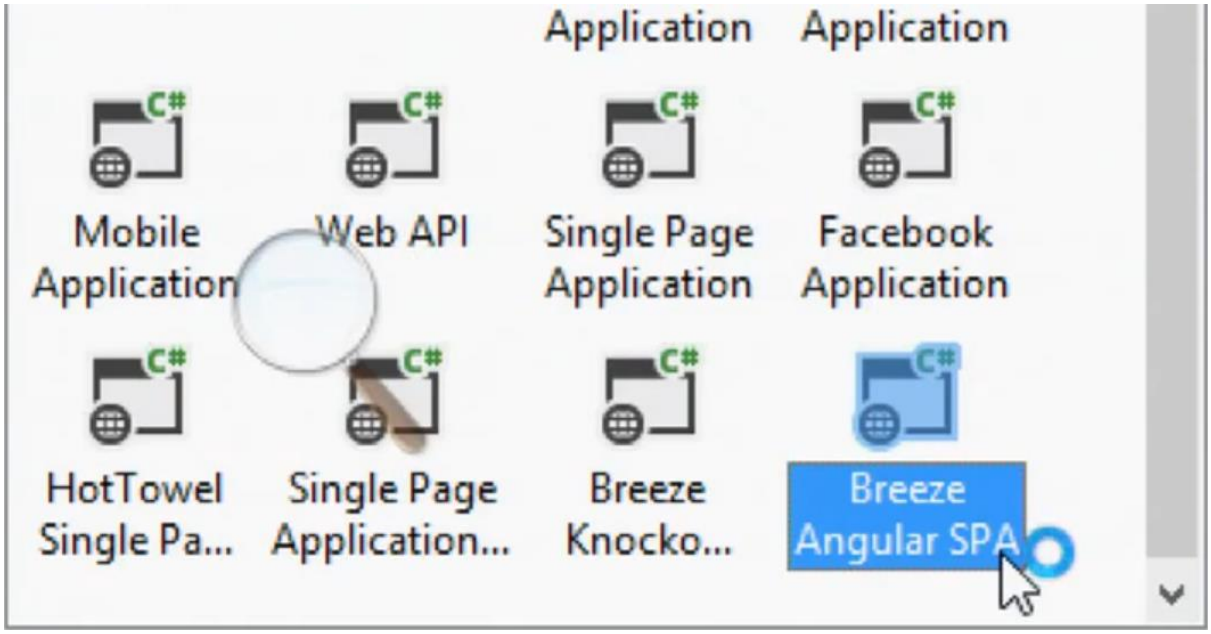
Demo: Angular and Breeze “To Do” App

I’ll spin up a new version of Visual Studio – that’s the IDE – and I wanted to create a new application I can do these template kind of things. There in the web I can see “MVC” and I can do that.

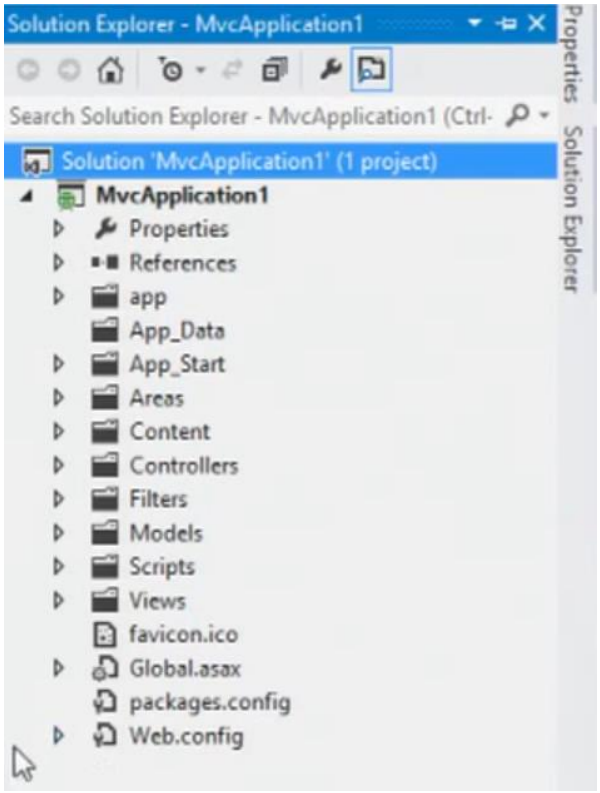




Ooh! What’s this? Yes, there’s the Angular SPA right in there.

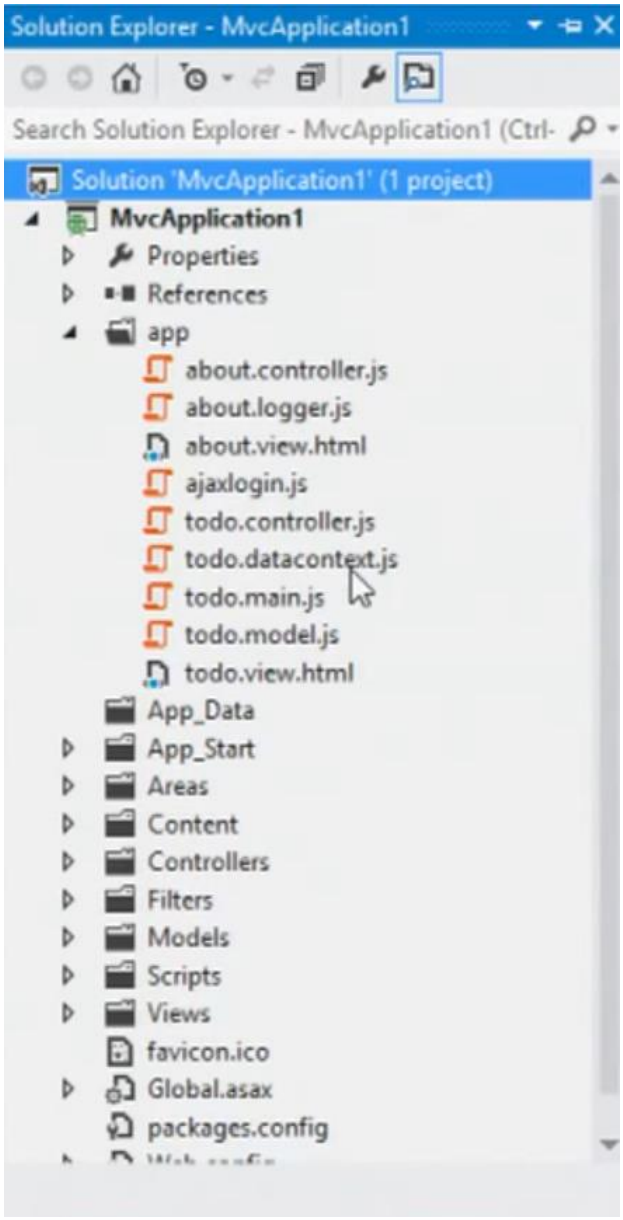


If I just keep on clicking I'm going to end up with an application that looks like this.

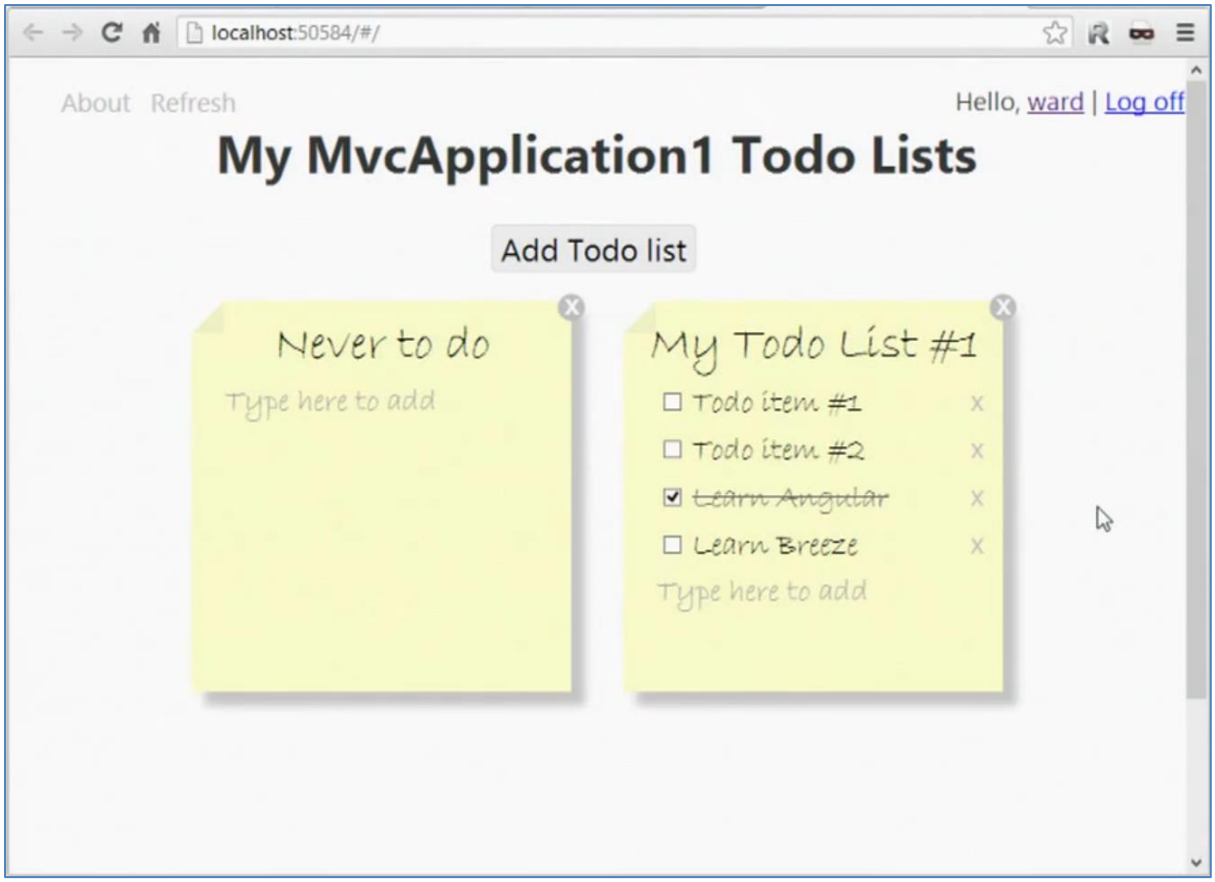


This is the ASP.NET MVC4 if you know that world.

There's a lot of stuff in there, but the nub of it is that your client-side application is right here in these JavaScript files and HTML files.



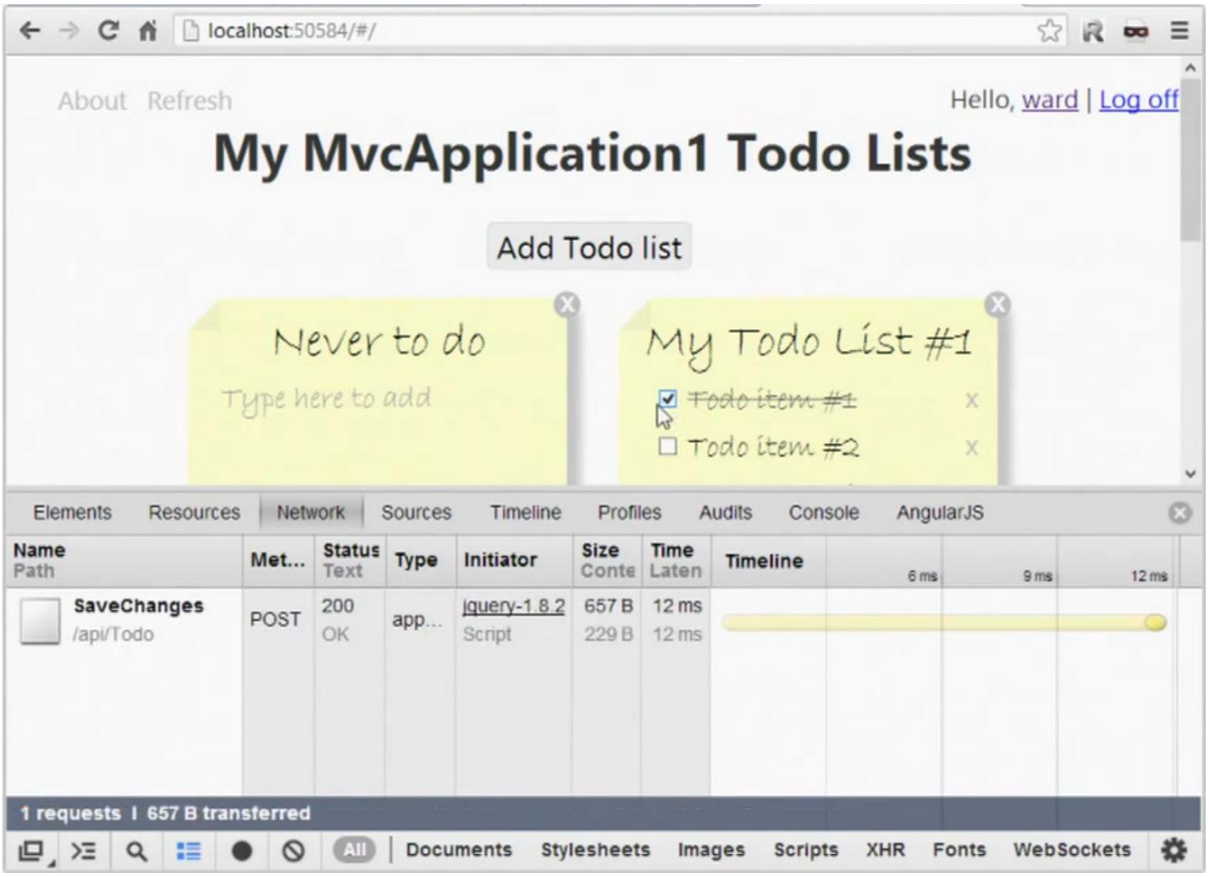
If I run it, it comes up:



09:05

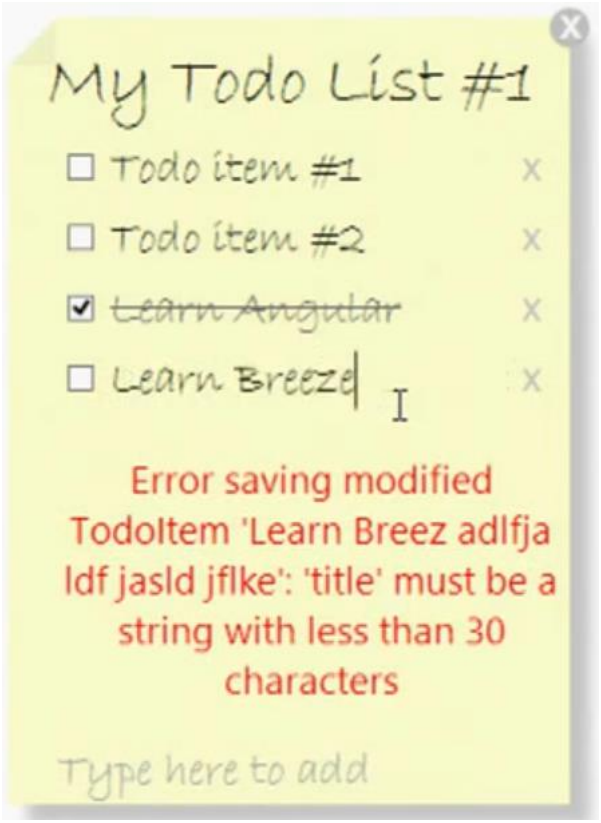
I've aligned some things because I've been here before. I've got my "To Do" list and my "Never to do" list and who knows what that's going to be. And here as you can see I've learnt Angular. I'd better learn it again.

Any time I'm going anywhere in here you'll see that it's saving….

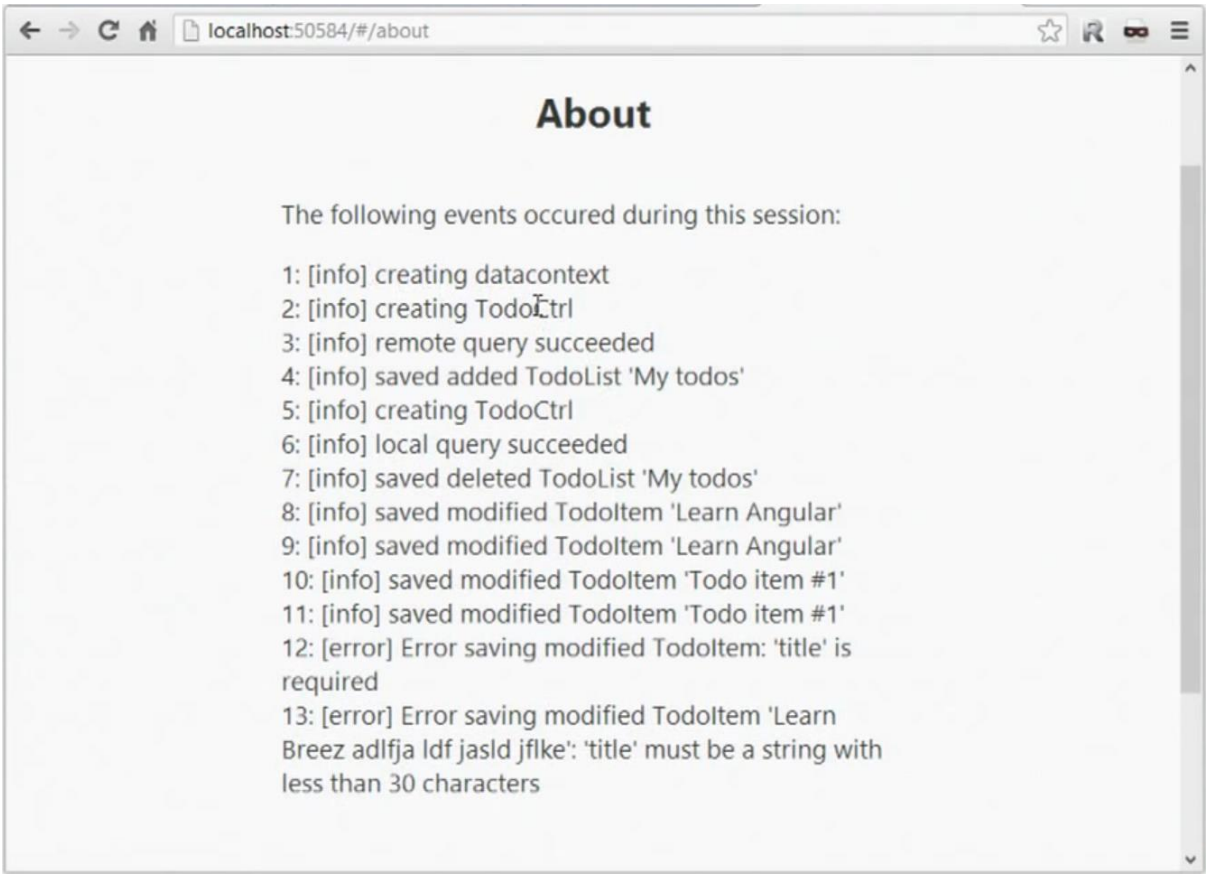


And if I say new item [no title entered] you can see there's validation:

… of all kinds…



I can switch over here on this second page [“About” link] and see what I did:

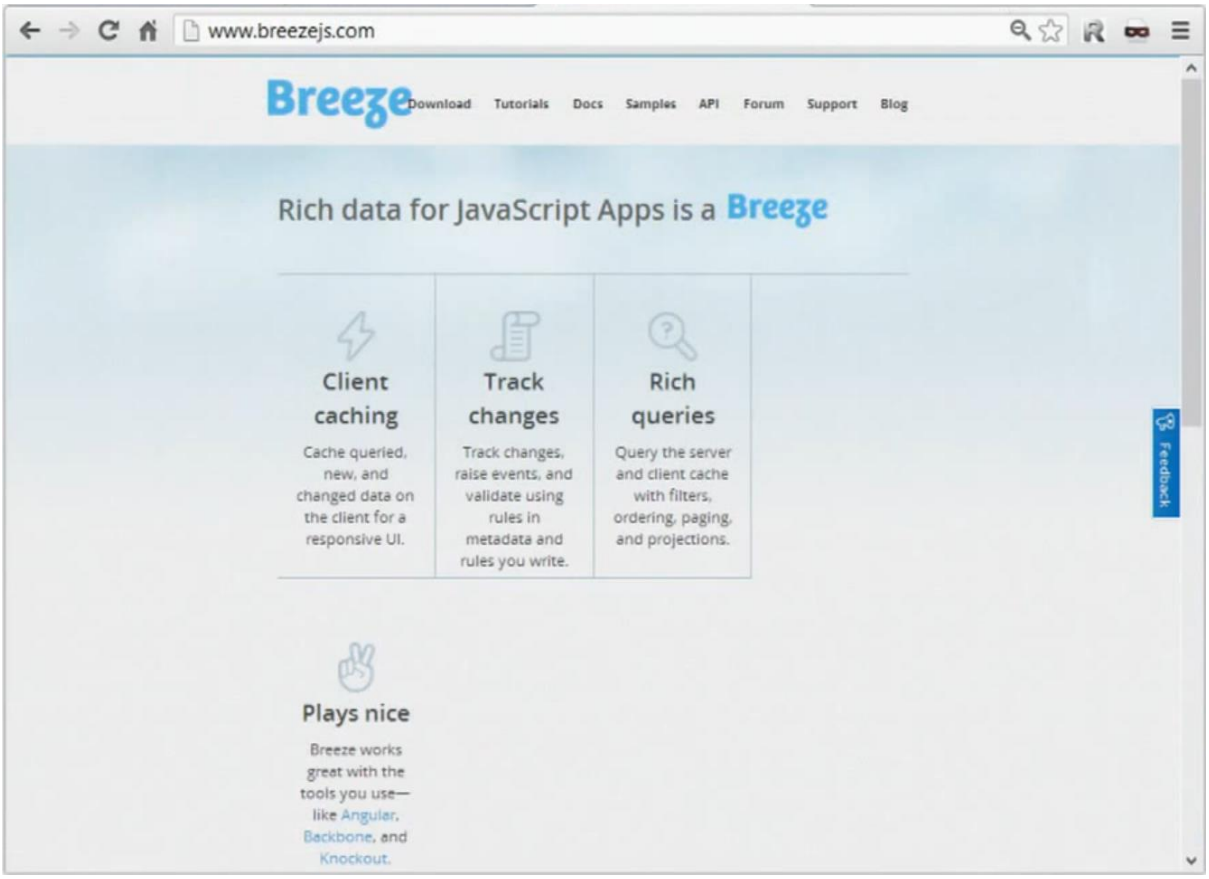


Really the second page is only about logging what's going on. You can see it's saved all these modified things so it has some idea of change tracking and error handling and it's all reporting that. You'll notice that it was initially going remote, but now it's going locally whatever the heck that means.

But that means that it could be kind of fast if I flick backwards and forwards [through the app] like this, because it's not making a trip to the server.

That's kind of what you want to do.

Angular and Breeze From Zero

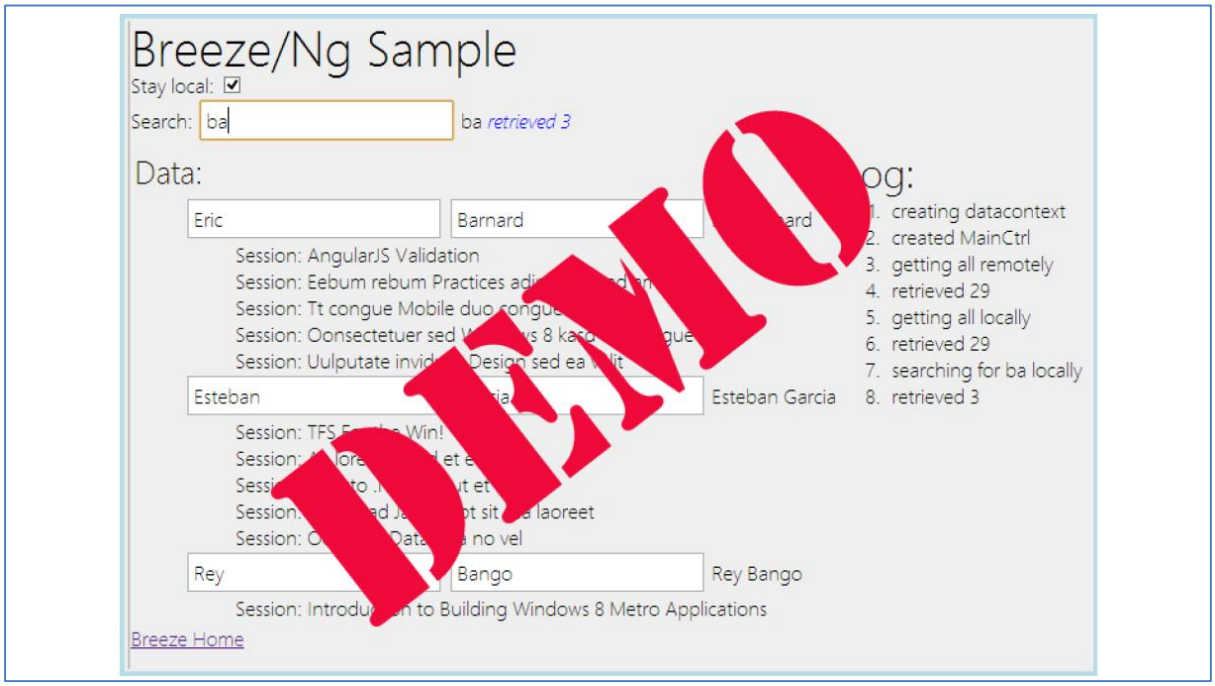


10:15

You can play with that to your heart's content, but I'm not going to do that anymore because we're going to take a trip through Angular and Breeze from what I hope is zero.

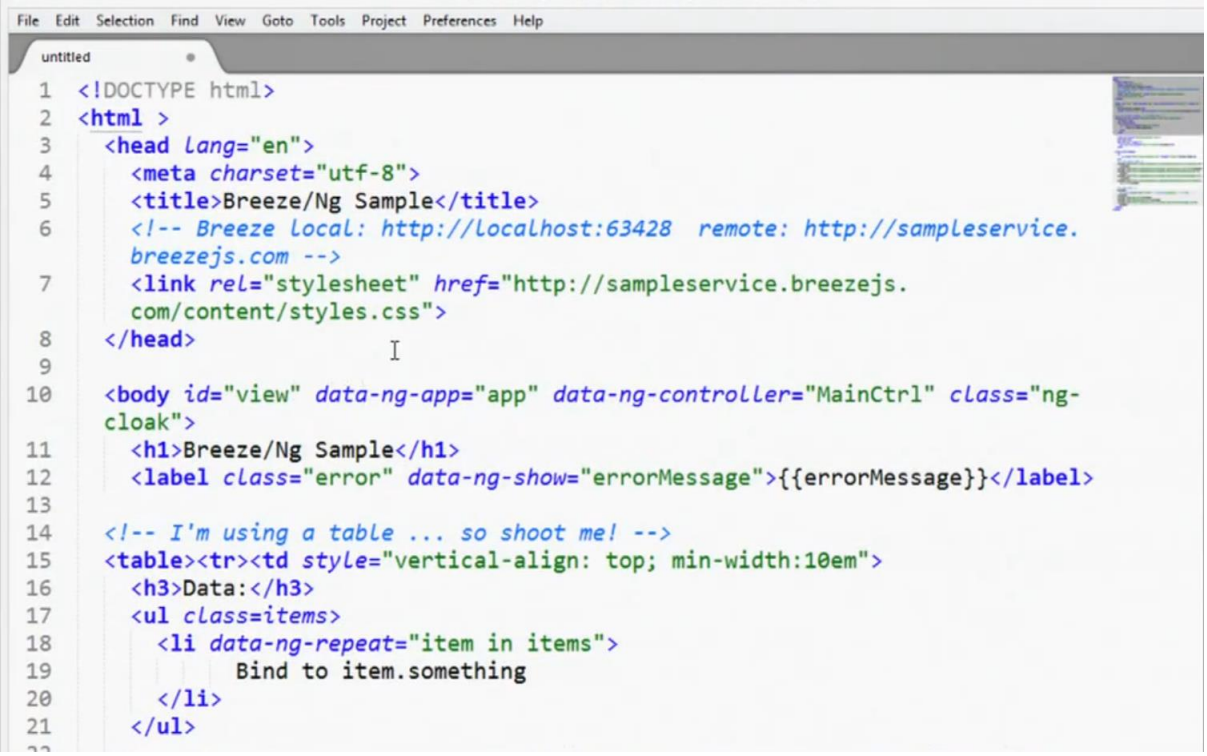
I'm also not using Visual Studio any more. From now on we're going to go with **Sublime**.

Building a Demo App with Sublime



10:35

I'm going to go with a new file here and **ngicreate**.

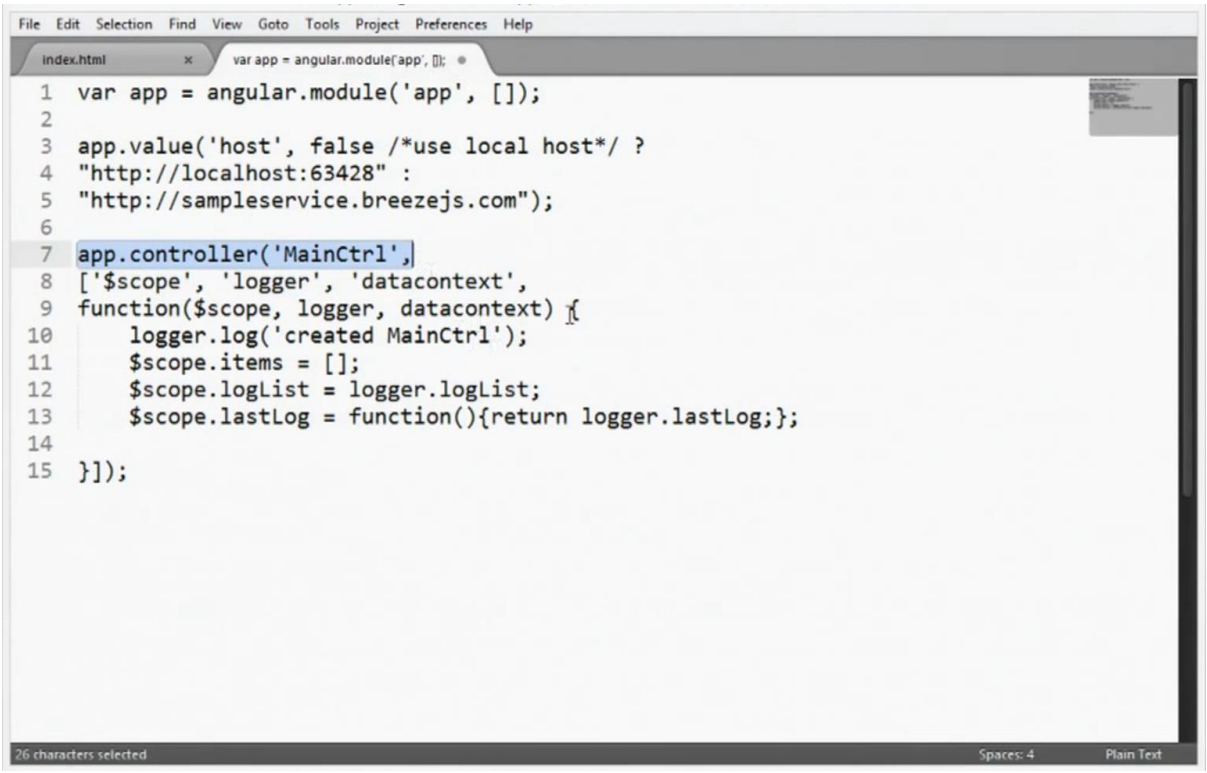


I am a really fast typist as you can tell. I'll spare you all of that.

This is just a big version of what Brad was showing during the warm-up. So there's an ng-app and a controller [MainCtrl] and there'll be some repeating and who knows what, but we'll just save that for now and we'll call that index.html and trust me we'll get to know it.

Let's create another one.

I have to have a controller. You saw that I mentioned the controller so I'll type really fast again and there is my controller….



11:30

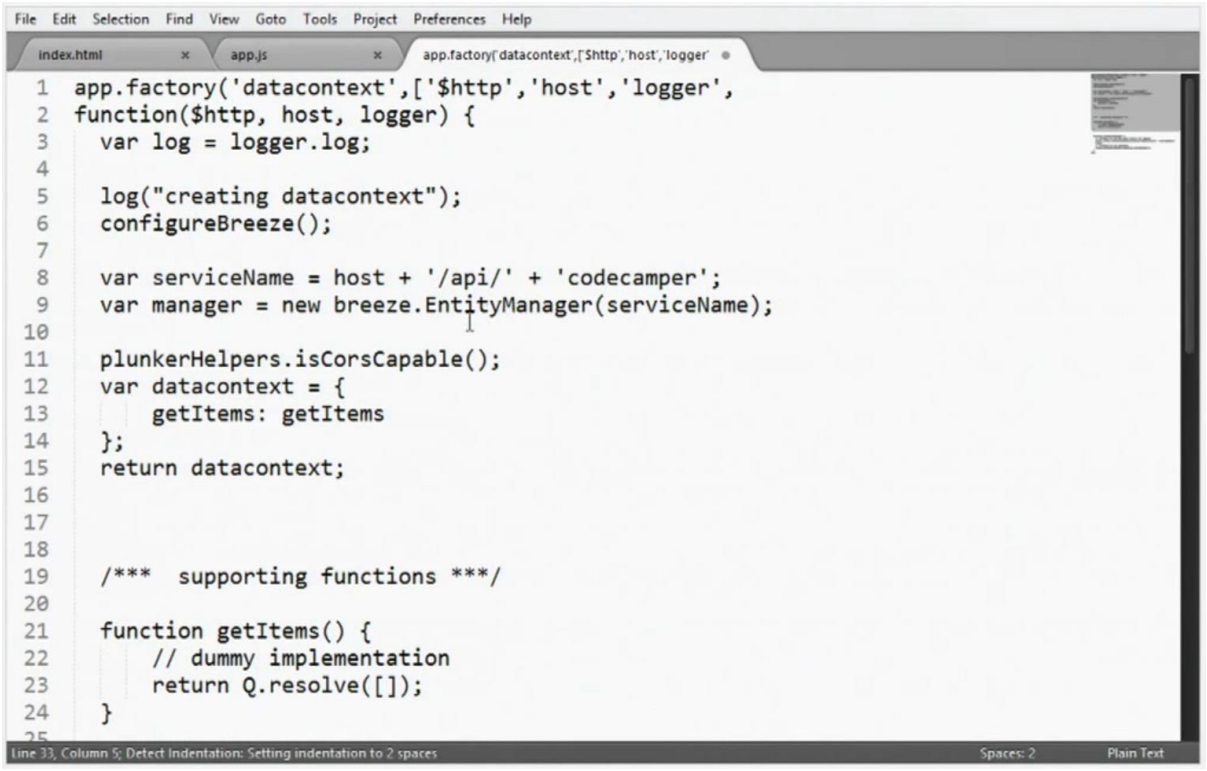
I've got a module. I'm stuffing the value in here. But this [highlighted in light blue] is really my controller here.

This [below the highlighted line] is dependency injection. I told you I would use everything in my Angular toolkit and I will. So I'm injecting some stuff here and that's the way it is but I'm going to expose a list of items and a logger and stuff like that.

We'll call that **app.js**.

So that's the controller side and my view side but one of my rules is you never do data access inside a controller or a viewmodel. The controller's job is to manage the view – it is not to go into the intricacies of accessing data so I always create some other class, a wrapper around it – that's good practice.

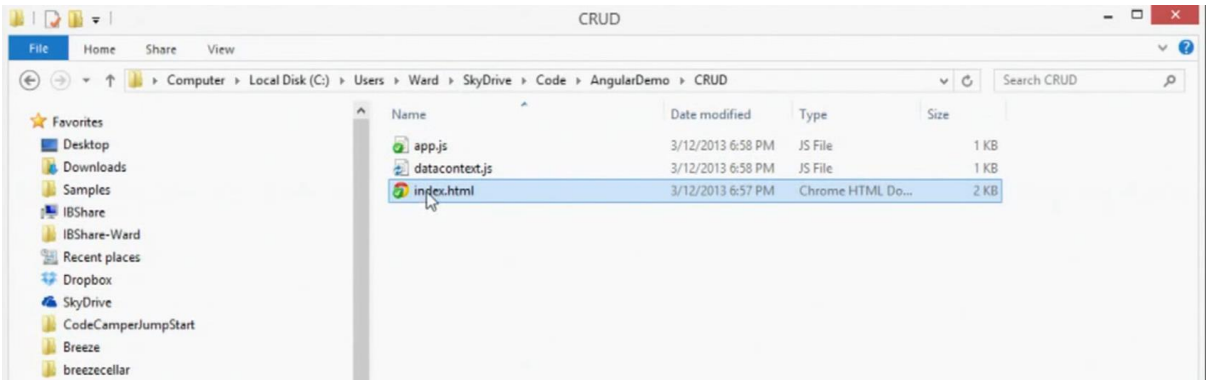
We'll call that our **datacontext** for historical reasons and wow! I typed fast again…



12:25

… and there it is.

Once again I'm using dependency injection for various reasons to get stuff in. I won't go into it because all this code will be made available to you but let's see what it does. I'm going to call it **datacontext.js** and if I have done everything right I will be able to go here…



.. and there's my index.html and …

… I guess it worked.

So how lucky is that? It worked the first time thank you very much. Now we can go home!

It's actually not showing very much.

First of all it's not showing anything because our controller's not going to get anything. It would be nice if it did so let's have our controller do that. This would be the kind of thing…

```
7  app.controller('MainCtrl',
8  ['$scope', 'logger', 'datacontext',
9  function($scope, logger, datacontext) {
10     logger.log('created MainCtrl');
11     $scope.items = [];
12     $scope.logList = logger.logList;
13     $scope.lastLog = function(){return logger.lastLog;};
14
15 getItems();
16
17   /***  supporting functions ***/
18
19   function getItems() {
20     datacontext.getItems()
21                .then(success)
22                .fail(failed)
23                .fin(refreshView);
24   }
25   function success(data) {
26       $scope.items = data;
27   }
28   function failed(error) {
29       $scope.errorMessage = error.message;
30   }
```

… where we get the items when the controller is instantiated. What it's going to do remember I always go outside: I'm not going to do any data access inside.

```
19    function getItems() {
20      datacontext.getItems()
21              .then(success)
22              .fail(failed)
23              .fin(refreshView);
24    }
25    function success(data) {
26        $scope.items = data;
27    }
28    function failed(error) {
29        $scope.errorMessage = error.message;
30    }
31    function refreshView() {
32        $scope.$apply();
33    }
34  }]);
```

I'm going to ask the datacontext to "Go get me some items" and because this is the way it is it's got to be asynchronous, right?

So this is the grand world of promises.

If you don't know what a promise is let me give you the intuition: I'm going away to go do something. I will give you a promise. When I come back I'll tell you whether I succeeded or not. If I succeeded then I'm going to call your **.then** callback. If I failed I'm going to call your **.fail** callback. And no matter what happens I'm going to call **refreshView** - **.fin** = finish.

So that's kind of the syntax for dealing with this **datacontext**. I don't know how it's going to do it but it's going to give me a promise and then I'm going to react.

And when I get it back I'm going to dump the data into the items so that they'll bind.

```
25    function success(data) {
26        $scope.items = data;
27    }
```

If there's an error I'm going to print it into the screen.

```
28    function failed(error) {
29        $scope.errorMessage = error.message;
30    }
```

No matter what happens I'm going to call this thing called "apply" which you do when you're doing asynchronous operations in Angular to tell it "Go refresh yourself".

```
31    function refreshView() {
32        $scope.$apply();
33    }
```

So we do that and my bet is that if we go and hit Refresh we won't get anything new:

If we look at the **datacontext** you will see that the getItems is returning nothing: a big nothing!

```
21    function getItems() {
22        // dummy implementation
23        return Q.resolve([]);
24    }
```

Let's make it return something…

Lo and behold, I'm going to get something.

```
21  function getItems() {
22      // dummy implementation: bind to {{item.name}}
23      return Q.resolve(
24              {results:[
25                  {LastName:"Minar", FirstName: "Igor"},
26                  {LastName:"Black", FirstName: "Naomi"}]
27              })
28              .then(getSucceeded)
29              .fail(getFailed);
30  }
```

What are we doing? Well I'm going to fake the data in here so I'm going to return some results of two people named "Igor Minar" and "Naomi Black".

One of the great things about promises is that you can chain them so I can do **.then**s and **.fail**s before the caller gets his **.then**s and **.fail**s.

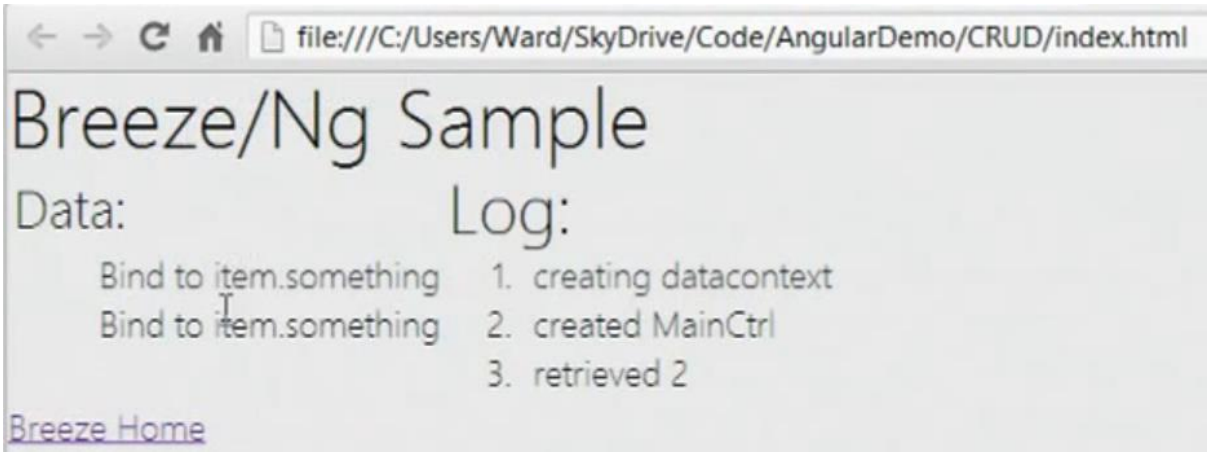
What I'm going to do when I succeed is I'm going to say "yahoo" and then I'm going to return something:

```
32  function getSucceeded(data) {
33      log("retrieved " + data.results.length);
34      return data.results;
35  }
```

Then if I go "Wah! Wah! Wah! Sad face" because I failed then that's what I'm going to do:

```
36  function getFailed(error) {
37      log("query failed: " + error.message);
38      throw error; // so caller can hear it
39  }
```

So we'll save that and let's see if that makes any difference.



15:50

It did, but I didn't see anything happen here. But at least I have two of something!

So I guess it's time to go and learn about binding!

And there it is [in **index.html**]

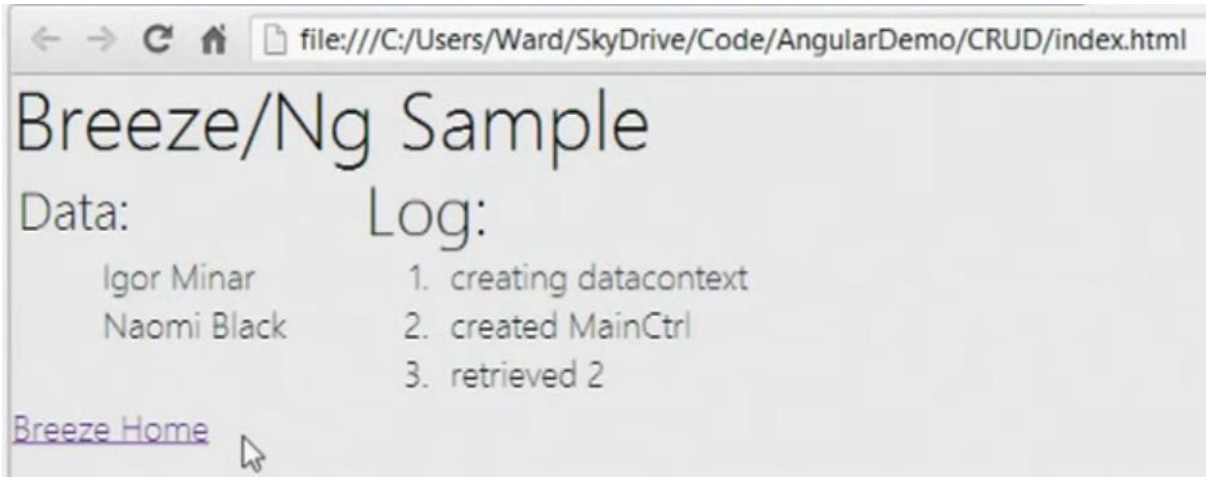
```
14  <!-- I'm using a table ... so shoot me! -->
15  <table><tr><td style="vertical-align: top; min-width:10em">
16    <h3>Data:</h3>
17    <ul class=items>
18      <li data-ng-repeat="item in items">
19            Bind to item.something
20      </li>
21    </ul>
```

Let's replace that:

```
17      <ul class=items>
18        <li data-ng-repeat="item in items">
19            {{item.FirstName}} {{item.LastName}}
20        </li>
21      </ul>
```

So for every item in items I'm going to show the first name, then a space, then the last name.

Let's see if it works… Heh! It does:



Yahoo! That's binding. Most of the time … you've seen this before.

That's very nice but most of the time we can't make an application from data that we store in the JavaScript.

Data Retrieval the $http Way

Let's do it the Angular way. Let's use that **$http** thing which is kind of a primitive way to do AJAX calls.

```
22  function getItems() {
23
24      var ngPromise = $http.get(serviceName+"/speakers");
25
26      return Q.when(ngPromise)
27              .then(getSucceeded)
28              .fail(getFailed);
29      }
```

16:50

I'm going to do this whole service thing. I'm going to get me a URL. I'm going to get back an Angular promise.

An Angular promise isn't quite like the promises I use in ***Breeze***, which are **Q.js** promises but **Q** can convert them for me and then I'll return a Q promise and let's just ignore the man behind the curtain and then you get to **getSucceeded** and **getFailed** and so forth.

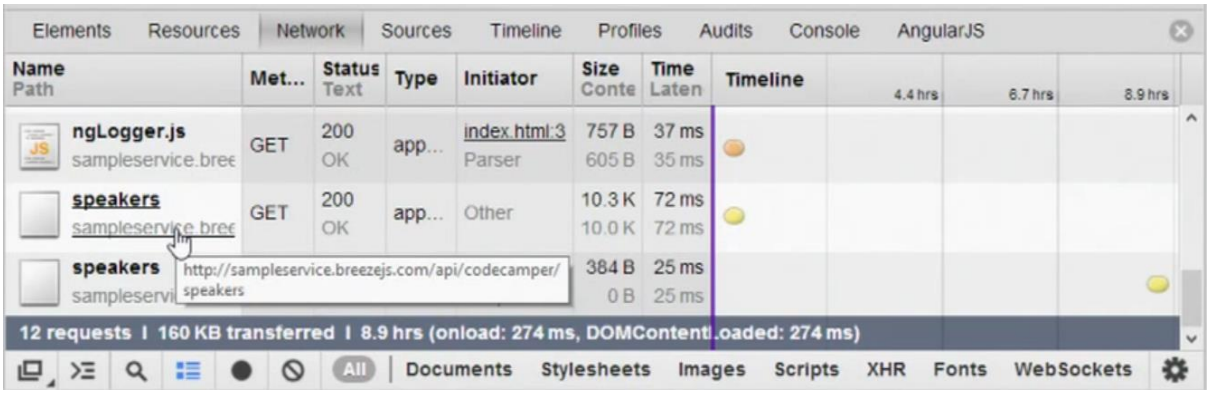
If I do this and I run it again I get an error message which is really not what I expected!

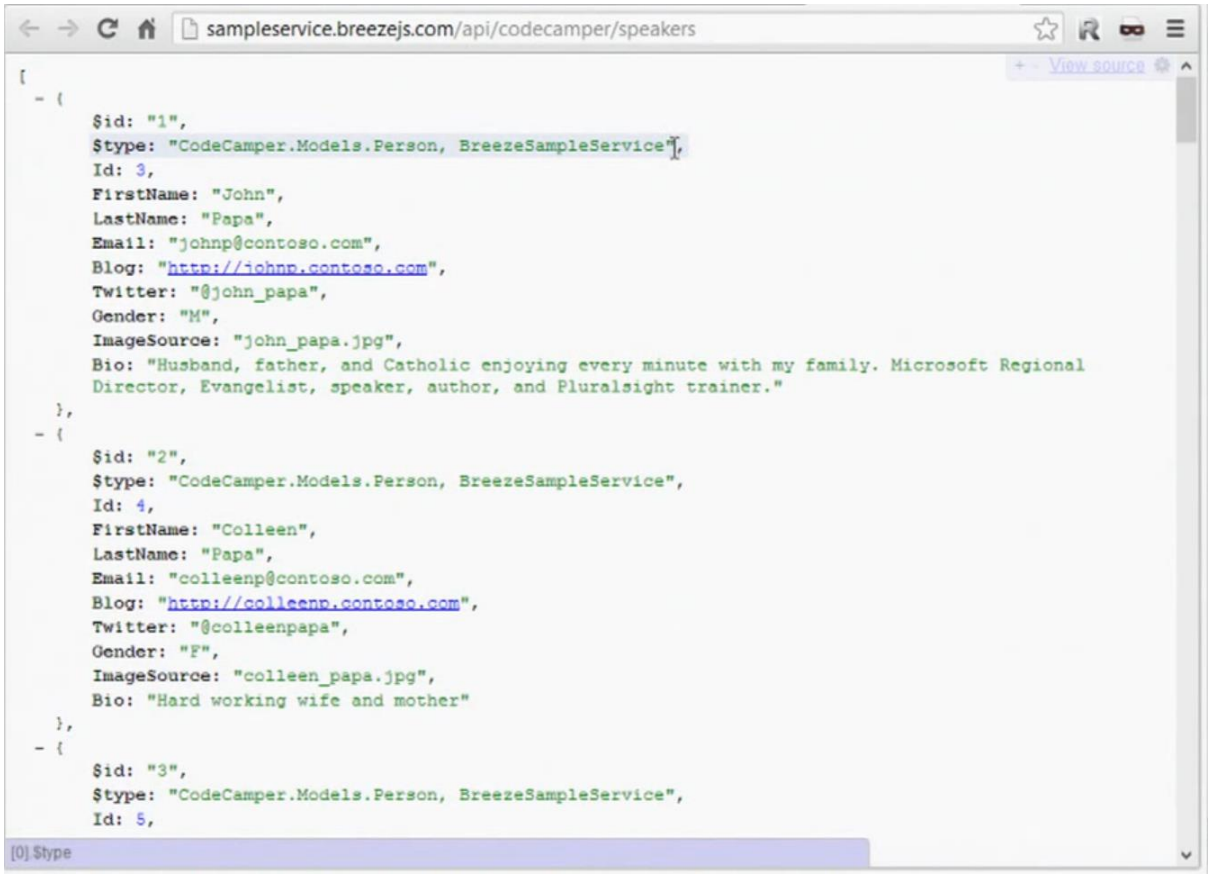
[Cause was duplication of two "function getItems()" lines through a cut and paste error]

Thank you audience [removes extra line]. Now I get what I want….

17:40

OK. How about that? I actually just made a trip out to a server somewhere and I got speakers from **sampleservice.breezejs.com** which you can go out to because it's just out there to give you some data to play with and that's what I got.





The whole idea was to pinch the first name and last name out of it.

That's good. But look at this [the original display screen] – they're in whatever order they're sitting in the database. You're not going to get a million dollar application that way.

So you're off to the races building your listing ship because you're $http handling all of your data access stuff and that's what people start off doing.

We're not going to do that because we're going to do it the ***Breeze*** way!

Data Retrieval the Breeze Way

```
21  function getItems() {
22
23  var query = breeze.EntityQuery.from("Speakers");
24
25    return manager.executeQuery(query)
26          .then(getSucceeded)
27          .fail(getFailed);
28    }
```

18:35

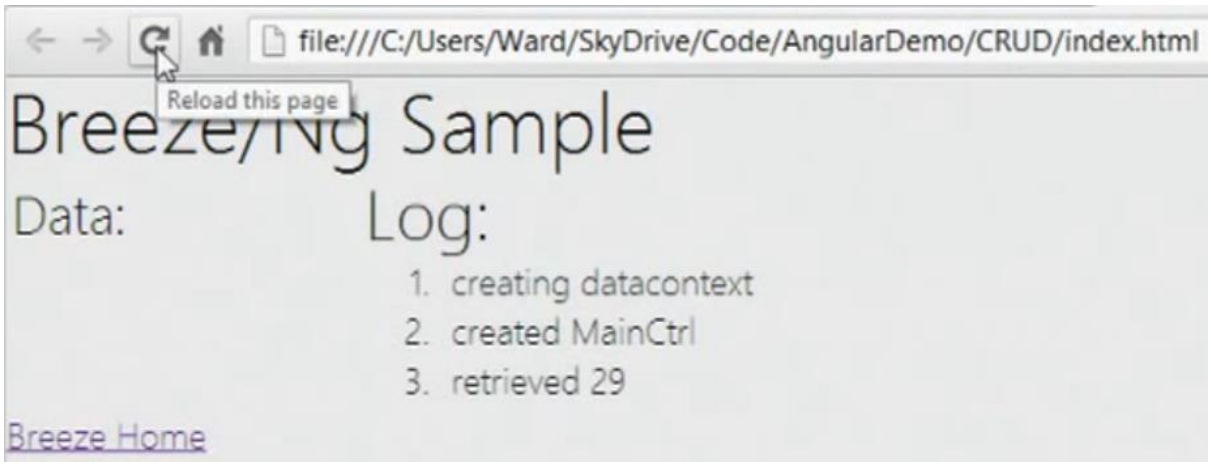
This time I'm using Breeze.

I'm going to create a new instance of this thing called an **EntityQuery**. It's a query object. I'm going to evolve this query object as we go but we're going to get it from the **Speakers** resource.

When we get this query we're going to get it back and then we're going to execute it with something called a **manager** which is your gateway to data and also a cache of entities.

Everything else is the same.

Now if I run it again… it doesn't work:



Why is that? The reason is because I actually can configure ***Breeze*** to say "so you know what, I don't like… in server land they're in Pascal case: first name with big "F" and so forth. But in JavaScript you like to have lower case so we do camel case". So we do some mapping things on the way in if you tell us to, and I told it to [in **datacontext.js**]:

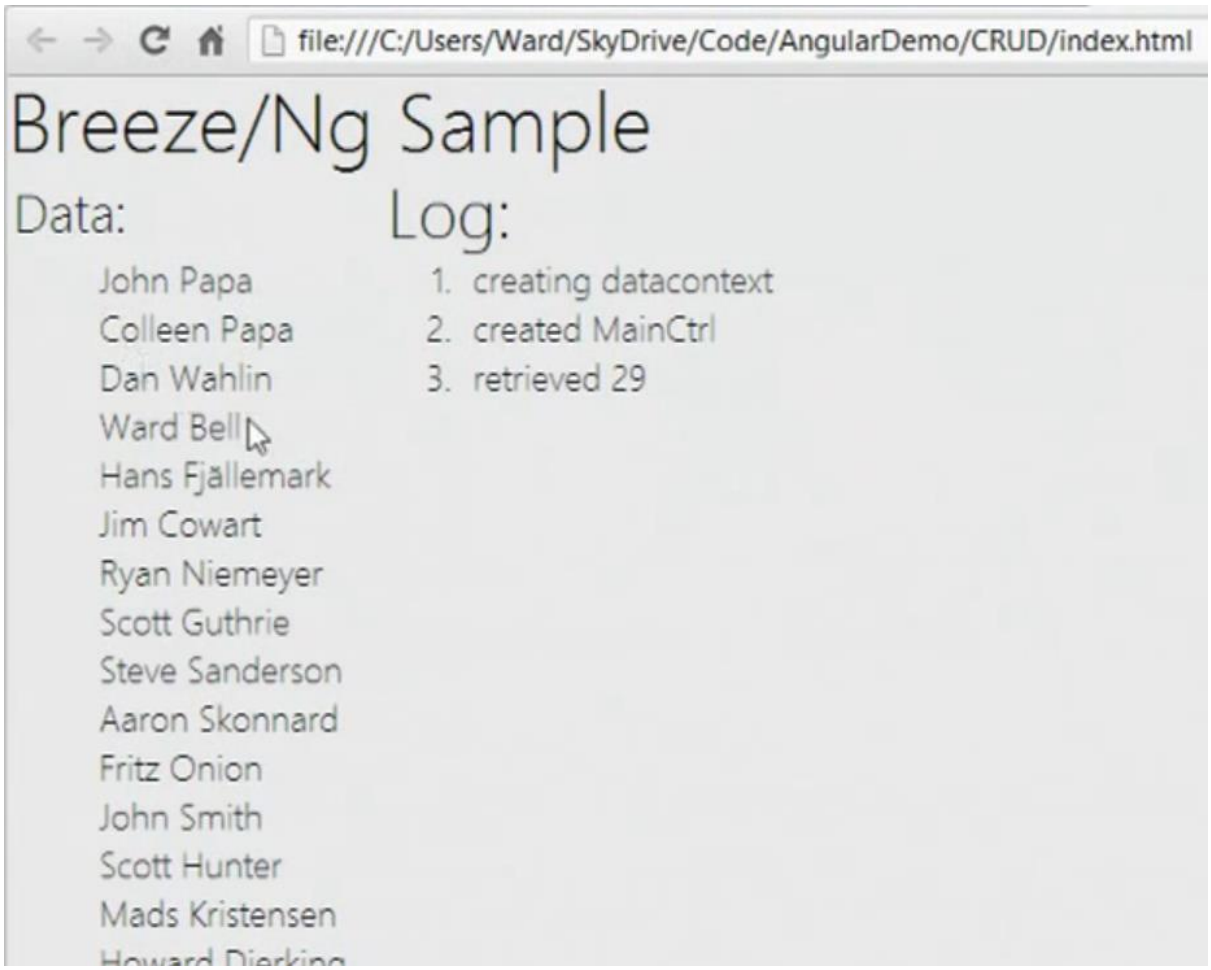
```
42  function configureBreeze() {
43    // configure to use the model library for Angular
44    breeze.config.initializeAdapterInstance("modelLibrary", "backingStore", tr
45    // configure to use camelCase
46    breeze.NamingConvention.camelCase.setAsDefault();
47  }
```

```
18      <li data-ng-repeat="item in items">
19          {{item.FirstName}} {{item.LastName}}
20      </li>
```

I'll just change that [index.html]:

```
18      <li data-ng-repeat="item in items">
19          {{item.firstName}} {{item.lastName}}
20      </li>
```

And maybe this time it will work…



Clearly something's a little bit different about the objects that are coming back. What's happening to them? They're actually coming back over the wire as JSON. We take those objects and we turn them into something called Breeze entities and we'll see that as it evolves. But for the moment they're just data bags like everything else.

Using Breeze to sort the Data

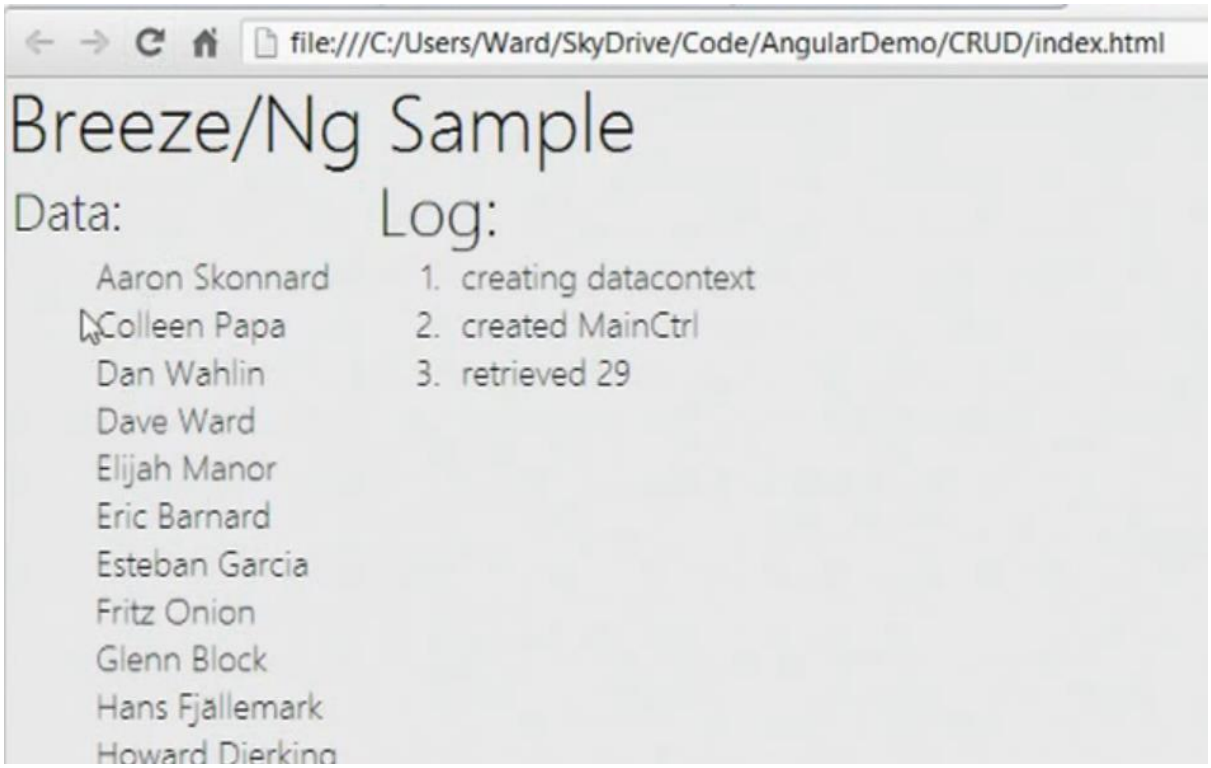
We still haven't really improved things because it's still in random order but we can start to do some interesting things all client-side. Like, for example – roll, please – I'm going to evolve this query object that we had here:

```
23  var query = breeze.EntityQuery.from("Speakers")
24      ;
```

I'm going to order this by first name and then last name:

```
23  var query = breeze.EntityQuery.from("Speakers")
24      .orderBy('firstName, lastName');
```

And if this works…. You can see that they are indeed sorted:



We got that sort.

Where did that sort take place?

It did not take place on the client. It took place on the server on the data tier.

It did not take place here. [It took place] On the data tier. That's very important!

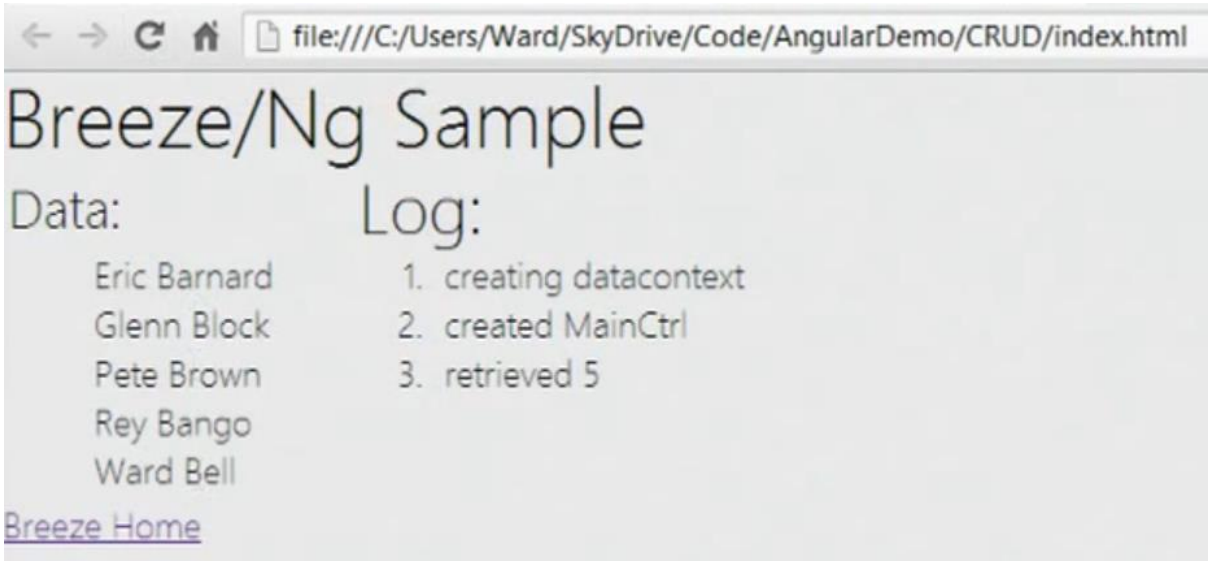
What else can we do?

Using Breeze to Filter Data

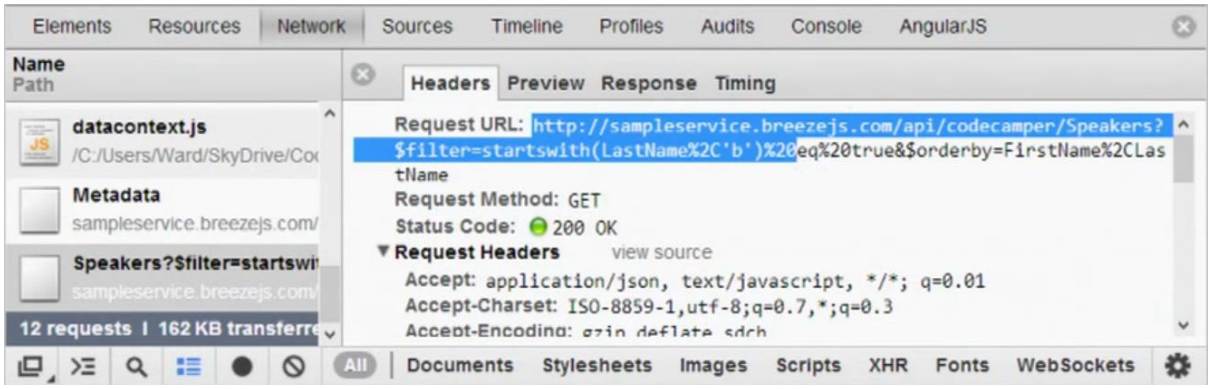
21:25

We obviously want to filter it. So let's say "last name" and there's a whole vocabulary here so it could be "starts with" and let's try "b".

```
23  var query = breeze.EntityQuery.from("Speakers")
24          .where("lastName", "startsWith", 'b')
25          .orderBy('firstName, lastName');
```



I'm posting all this data. It's kind of an OData query if you know OData:



So that happened on the data tier.

Let's see what else I could do there.

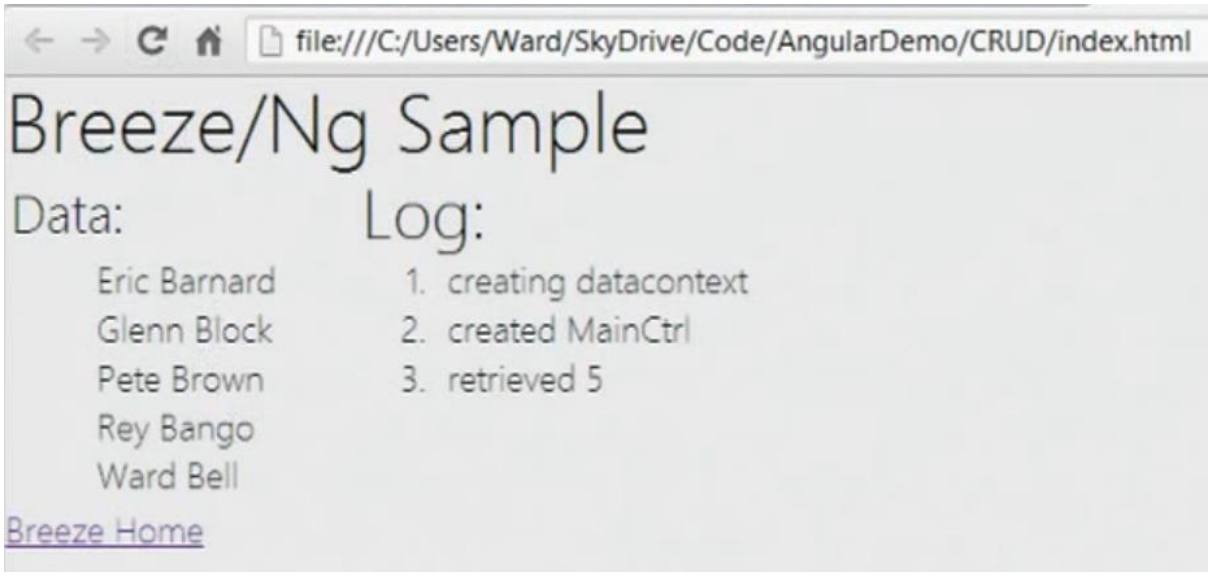
Reshaping the Data on the Client Side with Breeze

22:10

There was actually a fair amount of data that came back so I could say I really want to slim it. This [below] is called a **projection**.

```
23  var query = breeze.EntityQuery.from("Speakers")
24          .where("lastName", "startsWith", 'b')
25          .select('firstName, lastName')
26          .orderBy('firstName, lastName');
```

Let's see what that does for us:



No visible difference there except if we look at what came over it's very skinny: first name, last name, first name, last name…. none of the rest of the data that came over before:

So I can reshape the data on the client side.

Actually what I do is I shape my instruction on the client side and reshape it, send that instruction to the server, the server on the data tier does the reshaping and scales my data down.

You can't just sit there [with the where clause] plugging 'b's and 'c's and 'd's you've got to do what the user wants to do so let's put a search box in.

Adding a Search Box

23:20

Where shall we put it?

```
10  <body id="view" data-ng-app="app" data-ng-controller="MainCtrl" class="ng-
    cloak">
11    <h1>Breeze/Ng Sample</h1>
12
13    <p>Search: <input data-ng-model="searchText" /> {{searchText}}
14    <span data-ng-bind="lastLog()" class="message"></span></p>
15
16    <label class="error" data-ng-show="errorMessage">{{errorMessage}}</label>
```

There it is.

There's a little bit of binding to a thing called **searchText** which should be in our controller somewhere, and a bunch of other information, and …

Let's go add it to our scope. Our scope is getting bigger and bigger as we bind to things.

```
 9  function($scope, logger, datacontext) {
10      logger.log('created MainCtrl');
11      $scope.items = [];
12      $scope.logList = logger.logList;
13      $scope.lastLog = function(){return logger.lastLog;};
14      $scope.searchText = "";
15      $scope.$watch('searchText', getItems);
```

I'm going to add **searchText** and I'm going to add this other Angular thing called **watch**ing.

What I'm doing is I'm watching for changes in value. When I get a change in value I'm going to call **getItems**.

So every time I type something in the search box it's going to get detected and it's going to get passed to this **getItems**.

```
21  function getItems() {
22    datacontext.getItems()
23                .then(success)
24                .fail(failed)
25                .fin(refreshView);
26  }
```

That's all well and good but I'd better pass that information along because it probably needs the **searchText**!

```
21    function getItems() {
22      datacontext.getItems($scope.searchText)
23                 .then(success)
24                 .fail(failed)
25                 .fin(refreshView);
26    }
```

So I'm going to pass the search text to the **datacontext**.

That's good but the datacontext doesn't know about it yet.

Let's extend this query:

```
23  var query = breeze.EntityQuery.from("Speakers")
24        .where("lastName", "startsWith", 'b')
25        .orderBy('firstName, lastName');
26
27  //Add searchText as last param of getItems
28    // Both HERE and in APP.JS
29    if (searchText && (searchText = searchText.trim())) {
30      log("searching for " + searchText);
31      query = query.where('lastName', 'contains', searchText);
32    } else {
33      log("getting all");
34    }
```

I've got a little reminder here [the opening comment] so I don't mess up.

Again, **getItems** has to take a **searchText** parameter that's going to be the text I'm searching for:

```
21  function getItems(searchText) {
22
23  var query = breeze.EntityQuery.from("Speakers")
24        .where("lastName", "startsWith", 'b')
25        .orderBy('firstName, lastName');
26
27  //Add searchText as last param of getItems
```

When the search text comes in I'm going to evaluate it and trim it and all of that other stuff. If there's any search text left after I take off a whole lot of spaces then I'll say I'm searching for that.

But here's the real key. I'm building up the query [with the new .where clause].

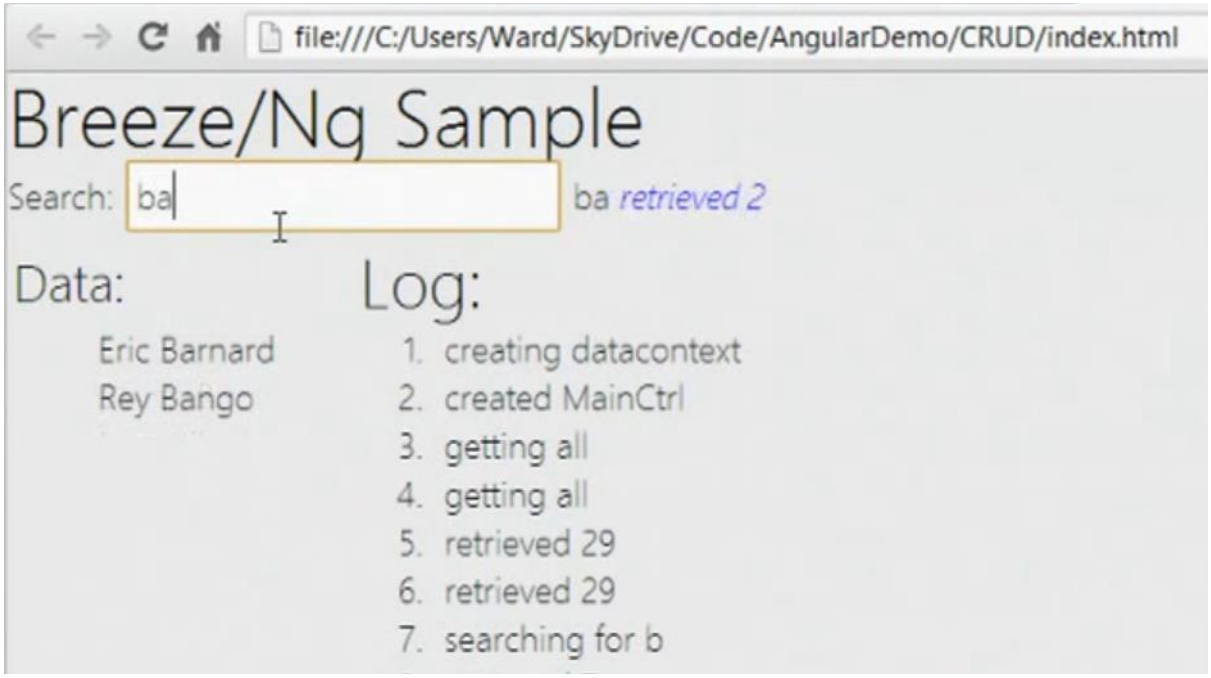
Actually I should get rid of that first where clause and get rid of that "b".

```
23  var query = breeze.EntityQuery.from("Speakers")
24        .orderBy('firstName, lastName');
25
26    if (searchText && (searchText = searchText.trim())) {
27      log("searching for " + searchText);
28      query = query.where('lastName', 'contains', searchText);
29    } else {
30      log("getting all");
31    }
```

If I have search text then I will add that filter clause on. If not I'm just going to go and get them all.

Let's see if it works.

Let's try "ba…".



How about that!

We're making progress here. That's looking closer to an LOB app.

Now what if…you know as long as I'm searching for last name why don't I search for anything?

I've got a text search so why not search for anything.

Let's see if we can do that…

Using Predicates with Breeze

```
23  var query = breeze.EntityQuery.from("Speakers")
24        .orderBy('firstName, lastName');
25
26    if (searchText && (searchText = searchText.trim())) {
27      log("searching for " + searchText);
28      var pred = breeze.Predicate
29          .create('lastName', 'contains', searchText)
30              .or('firstName', 'contains', searchText);
31
32      query = query.where(pred);
33    } else {
34      log("getting all");
35    }
```

26:15

What you can do is you can use predicates and you can build things up [using] **and**s and **or**s and all the kinds of strange expressions: you can build them up.

Again, I'm creating a client-side query. It's not running against the client – it's being sent to the server.

What I'm doing is I'm looking for last name or first name containing the search text. Will that help things any? Let's find out.

I have matches on "Barnard" [last name] and "Esteban" [first name]

We're getting closer to where I'm going here.

Actually the connection here is really great and apparently my server is going really great too so it's going really fast out there and this is not a realistic speed. We certainly have a load on it so I'm going to make this thing slow down by putting a delay of 1 second in there and then I'm going to do what I was doing before and execute the query.

```
37  return Q.delay(1000).then(function() {
38          return manager.executeQuery(query)
39              .then(getSucceeded)
40              .fail(getFailed);
41          });
42  }
```

I'm just making life a little bit slower and more realistic. And if I do that… I get an error message. That's not good. Anybody?

We're going to pretend it ran slower.

The point is if it were slow we would have an advantage to writing things locally vs going remote because we're going to be toggling back and forwards between pages as I was talking about earlier. So let's see if I can make that work.

We want this to be at the user's option. So in the HTML I'm going to put a little checkbox that binds to a **stayLocal** property so when I check it I'll stay local.

```
10  <body id="view" data-ng-app="app" data-ng-controller="MainCtrl" class="ng-
    cloak">
11    <h1>Breeze/Ng Sample</h1>
12    <p>Stay local: <input type='checkbox' data-ng-model="stayLocal" />
13    <p>Search: <input data-ng-model="searchText" /> {{searchText}}
14    <span data-ng-bind="lastLog()" class="message"></span></p>
15
16    <label class="error" data-ng-show="errorMessage">{{errorMessage}}</label>
```

Now it's got to bind to something, so:

```
16      $scope.stayLocal = false;
17      $scope.$watch('stayLocal', function(newVal, oldVal) {
18        if (newVal !== oldVal) {getItems(); }
19      });
```

So there's my **stayLocal** variable and I'm going to watch that guy too because when he fires up I'm gonna go "so first time I'm going to **getItems**".

What happens I'm going to end up here…

```
25    function getItems() {
26      datacontext.getItems($scope.searchText)
27                 .then(success)
28                 .fail(failed)
29                 .fin(refreshView);
30    }
```

.. except that I'm going to need to know or I'm going to need to tell him whether to stay local or not, so I'm going to pass that along to my **datacontext**.

```
25    function getItems() {
26      datacontext.getItems($scope.searchText, $scope.searchText)
27                 .then(success)
28                 .fail(failed)
29                 .fin(refreshView);
30    }
```

Now I'm actually doing it I'm saying to the datacontext "Heh. If you can go local I'm telling you to do it"

I guess I'd better do that, and pray for me now!

```
18
19    /***  supporting functions ***/
20
21  function getItems(searchText,stayLocal) {
22
23  var query = breeze.EntityQuery.from("Speakers")
24        .orderBy('firstName, lastName');
25
26  //Add stayLocal as last param of getItems
27    // in BOTH app.js and datacontext.js
28    var locally = stayLocal ? " locally" : " remotely";
29    if (stayLocal) {
30      query = query.using(breeze.FetchStrategy.FromLocalCache);
31      var locally = " locally";
32      var delayMs = 0;
33      manager.metadataStore.setEntityTypeForResourceName("Speakers", "Person");
34    } else {
35      var locally = " remotely";
36      var delayMs = 800;
37    }
38
```

I need to get that **stayLocal** thing in. I did that in both places so it's nice.

So what am I doing?

When the **stayLocal** comes in first of all make this little text variable [locally] so I can log the fact about whether I stayed local or whether I went remote.

If I'm stayling local notice the pattern here: I'm embellishing the query object again, and I'm going to say "Use the Breeze thing that tells me to query the cache"

It's going to be the same query – the exact same query I'd use to go remote, but I'm going to apply it to the cache of entities I already have.

Ignore the man behind the curtain there [reference to manager.metadataStore.setEntityTypeForResourceName in the code]

Otherwise I'm going to go remote.

All I did was I just decorated it with this little thing [FetchStrategy.FromLocalCache] and then I continued on as we did before.

```
37  if (searchText && (searchText = searchText.trim())) {
38    log("searching for " + searchText + locally);
39    var pred = breeze.Predicate
40        .create('lastName', 'contains', searchText)
41          .or('firstName', 'contains', searchText);
42
43    query = query.where(pred);
44  } else {
45    log("getting all"+locally);
46  }
47  // change Q.delay(1000) to Q.delay(delayMs)
```

Since we're not doing the delay anymore because I couldn't make that work I don't have to do that!

OK. Let's see what we got and let's go to the "Network" [tab in Chrome].

So the first time through I went and I got all the speakers.

Now I'm going to go local and let's see what happens now.

[Demo fails. Audience member points out that getItems fetches the searchText twice:

```
25  function getItems() {
26    datacontext.getItems($scope.searchText, $scope.searchText)
27               .then(success)
28               .fail(failed)
29               .fin(refreshView);
```

]

Corrected:

```
25  function getItems() {
26    datacontext.getItems($scope.searchText, $scope.stayLocal)
27               .then(success)
28               .fail(failed)
29               .fin(refreshView);
30  }
```

[Demo concludes by showing network traffic, selecting 'Stay Local" check box and then entering search text and Network tab showing no new network requests are being generated]

I can do this stuff as often as I like and I'm not making any round-server trips.

34:20

I talked about object graphs a bit earlier. You know Sessions have Speakers, and Speakers have Sessions. There's all this stuff and everything's connected but all I'm doing now is just getting one thing: Speakers – it's Persons.

That's not realistic. Beware the "single object" demo!

Breeze with Object Graph Data

So I'm going to get the sessions that are associated with each of these speakers. I could get them eagerly or I could get them dynamically as I need, but I'm going to get them eagerly by saying **expand**.

I need to know the navigation path from Speakers to Sessions and it's called **speakerSessions**.

```
23  var query = breeze.EntityQuery.from("Speakers")
24          .expand('speakerSessions')
25          .orderBy('firstName, lastName');
```

I'd like to be able to see that back in the HTML view so I'm going to replace my original list with this:

```
18  <!-- I'm using a table ... so shoot me! -->
19  <table><tr><td style="vertical-align: top; min-width:10em">
20    <h3>Data:</h3>
21
22    <ul class=items>
23      <li data-ng-repeat="item in items">
24          {{item.firstName}} {{item.lastName}}
25          <ul>
26            <li data-ng-repeat="session in item.speakerSessions">
27                Session: {{session.title}}
28            </li>
29          </ul>
30      </li>
31    </ul>
```

Let's take a second and see what this is.

I'm still saying <ul>. I've still got firstName, lastName, but it's got an inner repeater that's going to get the session information from the related sessions.

Pray for me!...

Heh! It worked – sort of!

If I stay local it's going really fast and I'm getting all this stuff.

OK. You get the idea? Outer one [list], inner one [list] and Breeze – simply because I said "When I get a speaker expand and I want to get those speaker sessions right along with it.

Of course I could do it down on the server. I could do it on the client. I have a lot of options as I'm developing my client-side app.

That's nice but I'm still dealing with essentially data bags.

By the way you guys can ask me a question if you like. I'm still dealing with data bags. The stuff that's coming over is Speakers – first name, last name, just property stuff.

We've seen the query thing. What can we do now that we have the entities in cache? There's more things that you need to do. It's not just about querying. It's about change tracking and validation and stuff like that.

You may have noticed that I never defined Speaker. Did you notice that I never went through and described what a speaker was and what its properties were? It just handled them.

That's a feature of ***Breeze***. One of the things it's doing is it's getting metadata to describe the model down from the server and it's using it so that you don't have to code it.

Defining the Model and adding Rich Behaviour using the Module Revealing Pattern

You've seen other systems. You've probably seen where you have to describe the whole model in some kind of a DSL. You can do that also if you want but you don't have to.

Let me create a model class and save it as **model.js**

So I don't forget we want to make sure that we load it.

```
55      <script src="app.js"></script>
56      <script src="datacontext.js"></script>
57          <script src="model.js"></script>
58      <script src="http://sampleservice.breezejs.com/scripts/ngLogger.js"></
          script>
59    </body>
```

Let me walk this for you a little bit.

39:25

```
1   /* model */
2   app.factory('model', function() {
3
4       extendModel();
5
6       var model = {
7           initialize: initialize
8       };
9
10      return model;
11
12      //#region private members
13      function initialize(metadataStore) {
14          metadataStore.registerEntityTypeCtor("Person", Person);
15      }
16
17      function Person() {
18          this.firstName = "Ima";    // defaults
19          this.lastName ="Noobie";
20      }
21
22      function extendModel() {
23          Person.prototype.fullName = function() {
24              return this.firstName + " " + this.lastName;
```

Well it doesn't do much. It's the revealing module pattern. It has an initializer. That's all it's got and it's got a thing that it does at the beginning called extend.

Next is the initialize which I'll show you in a second but first, I just want to show you that there's a Person – a Person constructor. If I want to create a constructor I can. I didn't have to before but I can.

One reason I might is that I might want to set some defaults like “Ima Newbie” if I create a new person.

Also I’d like this person to have some behaviour.

```
22      function extendModel() {
23          Person.prototype.fullName = function() {
24              return this.firstName + " " + this.lastName;
25          }
26          Person.prototype.isDirty = function() {
27              return !this.entityAspect.entityState.isUnchanged();
28          }
29      }
30      //#endregion
31  });
```

I can’t send behaviour down from the server. But I might like to have a **fullName** property and I might want to know if it’s dirty if a change is being made to this person. This [entityAspect.entityState.isUnchanged] is some crazy stuff that actually gets that.

So I’ve got that. Now I have to initialize it.

I’m back in the datacontext. This is my data access thing and I’m going to get the model using Angular dependency injection here. If I slow down it may actually work.

```
1  app.factory('datacontext',['$http','host','logger','model',
2  function($http, host, logger, model) {
3    var log = logger.log;
```

Now that I’ve got the model it had only one method on it which was an **initialize**. Basically I’ve just got to tell it about the metadata which I’m going to get from the **EntityManager**.

```
 8    var serviceName = host + '/api/' + 'codecamper';
 9    var manager = new breeze.EntityManager(serviceName);
10    model.initialize(manager.metadataStore);
```

So really what I’m doing here is there’s metadata stored that describes the model, all the relationships between the objects, and what they have and stuff like that.

What we’ve done is just evolved it with our own stuff [in function extendModel]. So you’re not restricted to just a data bag coming down from the server. You can make rich objects that have behaviour and properties and stuff. And all these can be serialized too if you want.

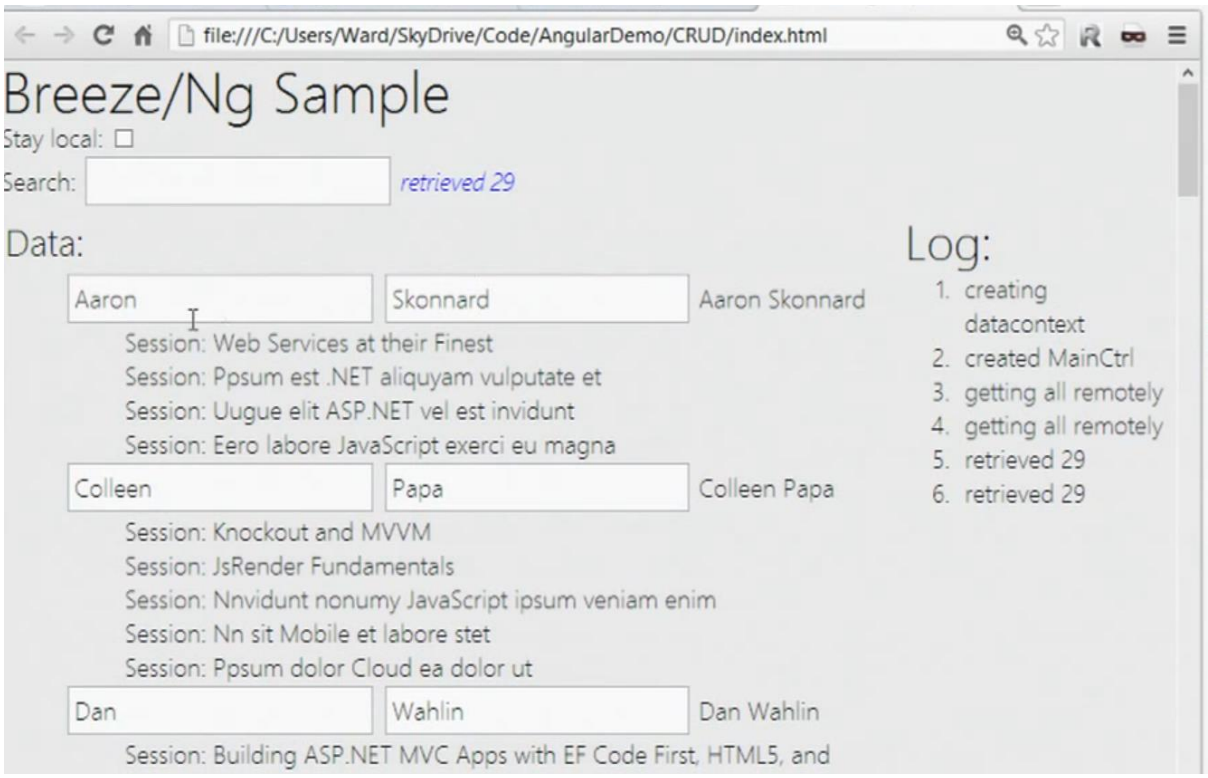
OK. So we’ve got that. Let’s kinda see how that works.

I’m going to go back to the HTML and I want to see it so to do that I’m going to use that **fullName** thing.

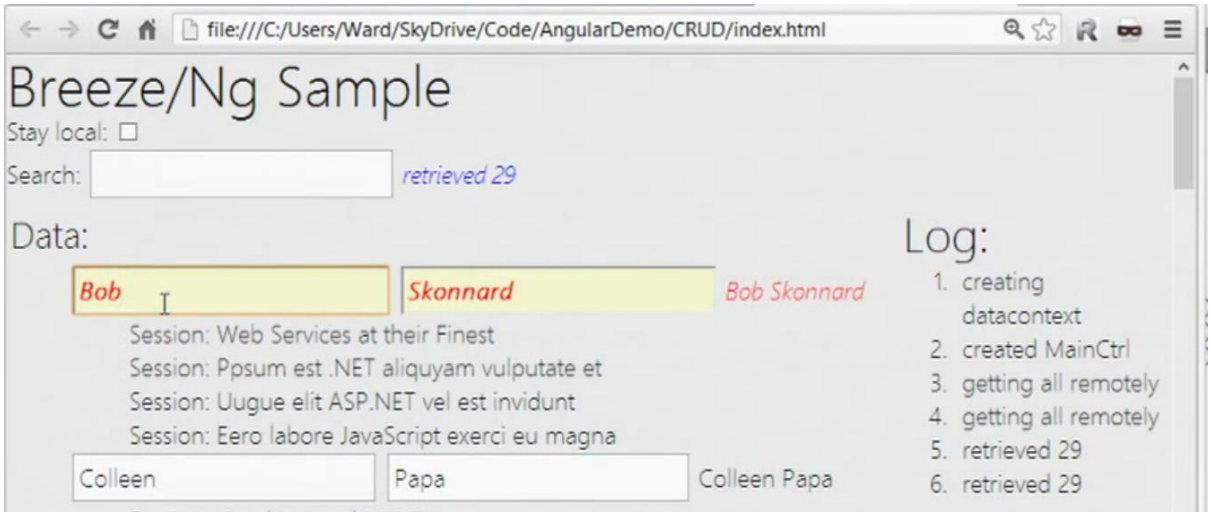
```
22    <ul class=items>
23      <li data-ng-repeat="item in items">
24
25          <input data-ng-model="item.firstName" data-ng-class="{isDirty:item.
            isDirty()}"/>
26    <input data-ng-model="item.lastName"  data-ng-class="{isDirty:item.isDirty
      ()}"/>
27    <span data-ng-bind="item.fullName()"  data-ng-class="{isDirty:item.isDirty
      ()}"></span>
28          <ul>
29            <li data-ng-repeat="session in item.speakerSessions">
30                Session: {{session.title}}
31            </li>
32          </ul>
33      </li>
34    </ul>
```

41:55

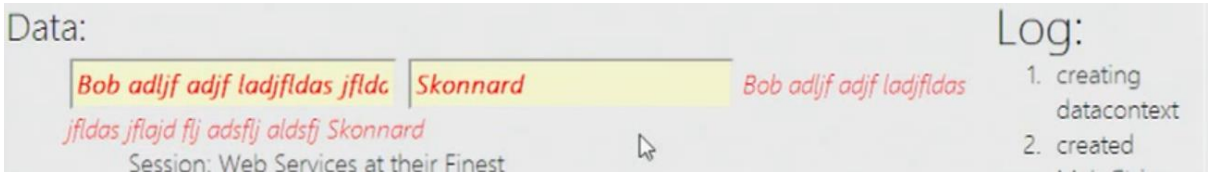
So I've got a first name input text box and a second name input text box and then a full name here:



As I make changes, if I call him [Aaron] Bob I have some indication that it's in an "isDirty" state. So I'm leveraging that as well.



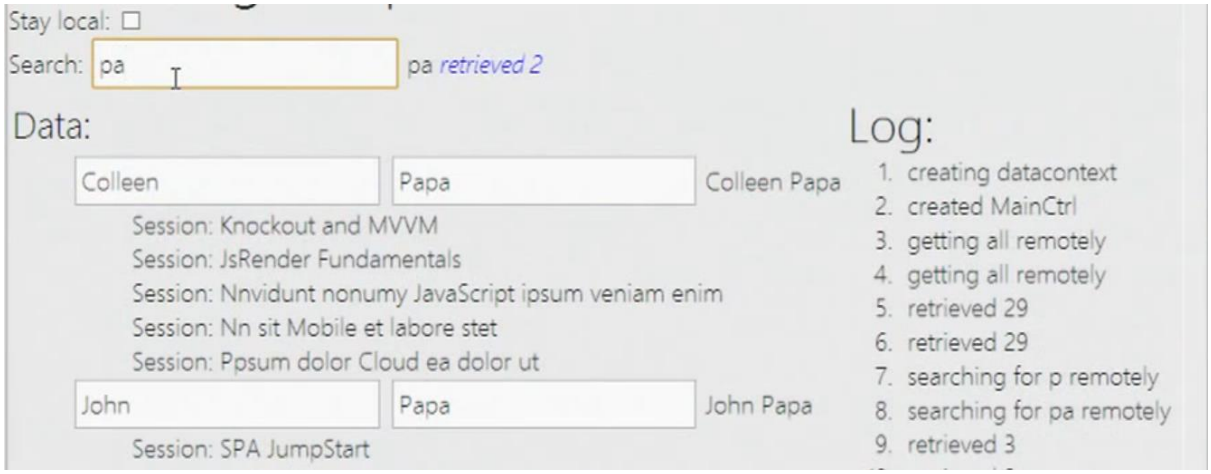
Now I'm going to do stuff like this which I know is illegal.



If I had given the ability to save this would be interesting.

The thing that's kind of cool here though is that this is dirty and there's all this stuff but this is not part of the form.

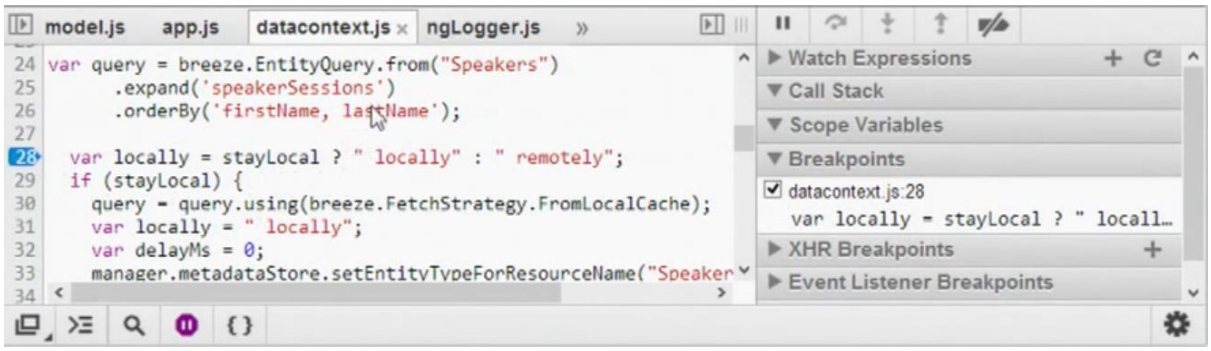
I just need to change this [search text box] and if I go look at "ba" it still works. That changed the screen right:



But if I come back, he's still there.

So we're talking about a property, a characteristic of the entity, not of the screen. The screen isn't dirty, the Speaker is dirty – a very important distinction.

Let's put a breakpoint in here and see what's going on behind the scenes. I'm going to do that where it does the "get locally" thing:

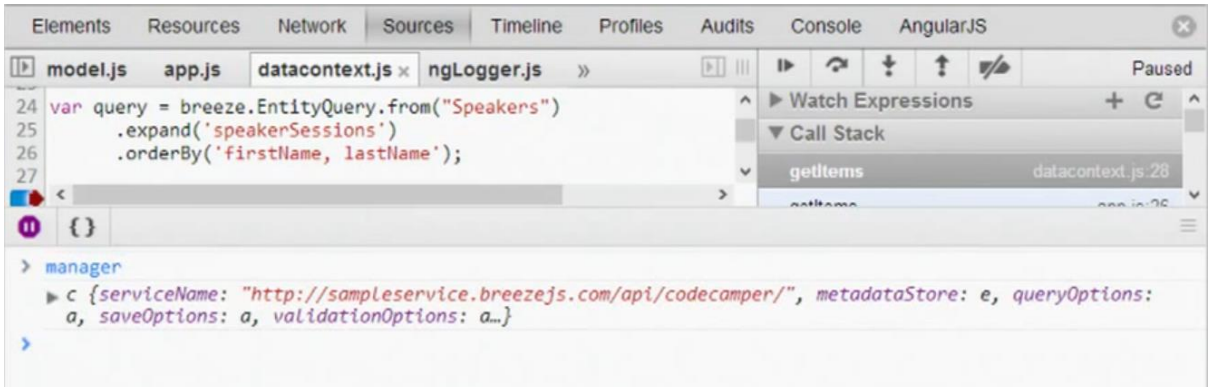


Let's do that [selects Stay Local]…

… and now we're stopped…

… and let's do a little spelunking of what we've got here…

So we have a manager, and we know about him.



We can ask the manager "Do you have changes?" and indeed we do…

I didn't have to say anything. It's accumulated changes. It'll accumulate changes to anything, any of the entities it's tracking. It could be Persons. It can be Sessions. It can be Rooms. It can be anything.

So let's take a look at the first one and just do a reality check…

Yup, there's Bob!

Yes, he has full name.

Using entityAspect to see what's going on under the covers in Breeze

45:25

There's a way to find out… there's an **entityAspect** property on every entity and that's your gateway into "entity-ness". Normally we try not to pollute the object. It should have the features of a Person on it. But every once in a while you want to get under the covers to see how is ***Breeze*** thinking about this thing and that's what the **entityAspect** lets you do.

So I can ask about the **entityState** because that's "entity-ness":

```
> manager.getChanges()[0].entityAspect.entityState
  ▶ c {name: "Modified", parentEnum: b, isUnchanged: function, isAdded: function, isModified: function…}
```

Yup, that's right – it's modified! That's what I kinda expected.

How about the original values?

```
> manager.getChanges()[0].entityAspect.originalValues
  Object {firstName: "Aaron"}
```

So we know that has changed, and what's changed.

I said that… I gave it a really ridiculous name, I wonder if that's a problem. Yes indeed, there is a validation error. The error message is there:

```
> manager.getChanges()[0].entityAspect.getValidationErrors()
  [▼ a                                                                    ]
     ▶ context: Object
       errorMessage: "'firstName' must be a string with less than 30 characters"
       key: "firstName:maxLength"
     ▶ property: a
       propertyName: "firstName"
     ▶ validator: k
     ▶ __proto__: Object
```

It actually does, as you make changes to a property, it immediately validates them. You could just disable that if you want. But that could be useful if you wanted your form to immediately light up with an indication that this was an invalid first name.

One thing's for sure, there are several times – one of the times is "save" so this thing will never try to save. If I try to save this back to the server it will refuse. It will refuse because it failed to pass client-side validation.

Now of course that's no substitute for server-side validation, but it gives you a better user experience to tell them that they screwed up on the client before you make that long trip to the server.

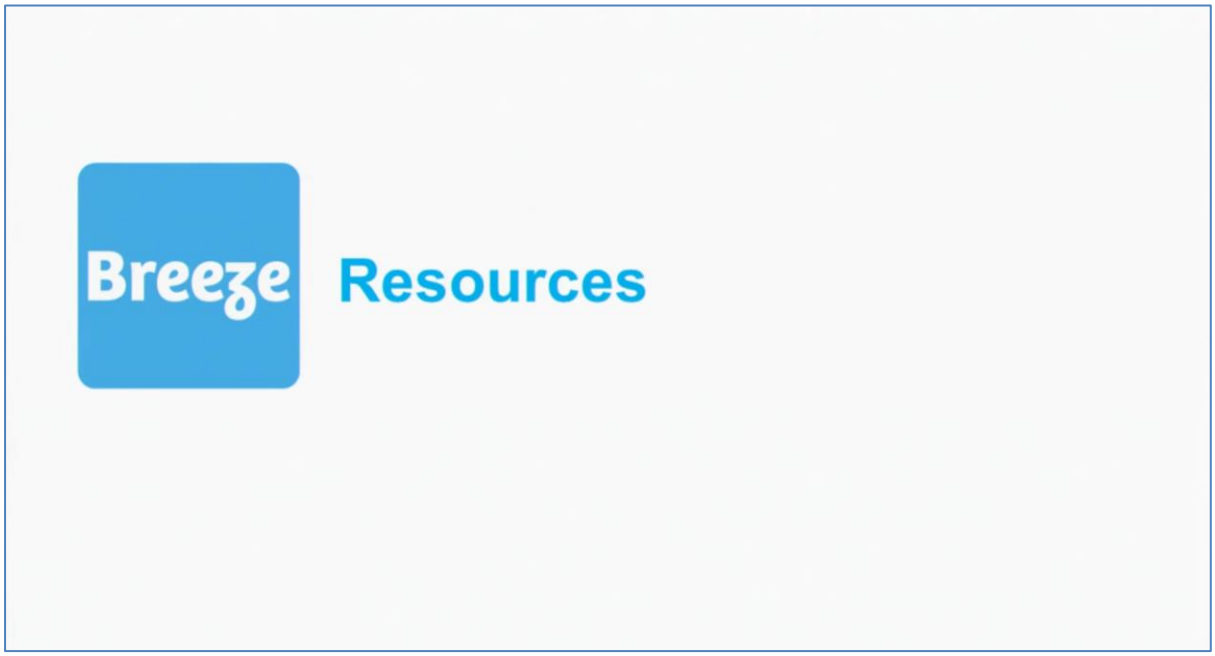
I can say "You know what? I don't like what's going on here. I want to roll it all back."

```
> manager.rejectChanges()
  [▶ Person ]
```

If I do that, you will notice that Aaron is right back where he was:

I think at that point I want to stop, bring it on home and then take your questions.

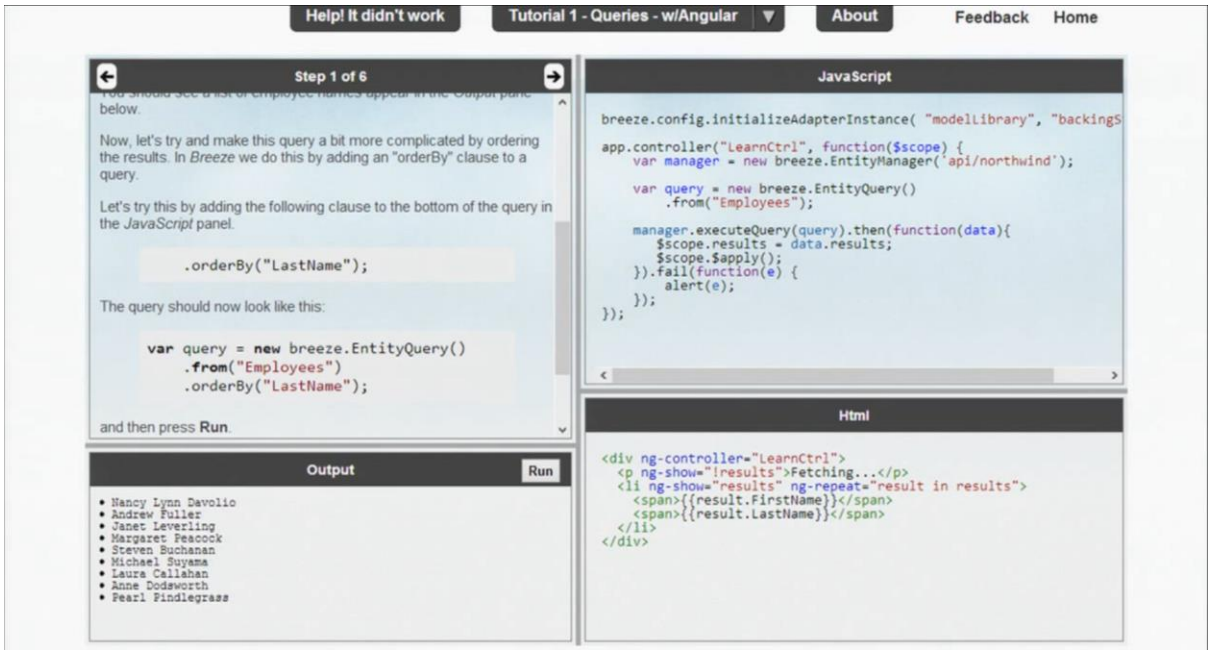
Breeze Resources

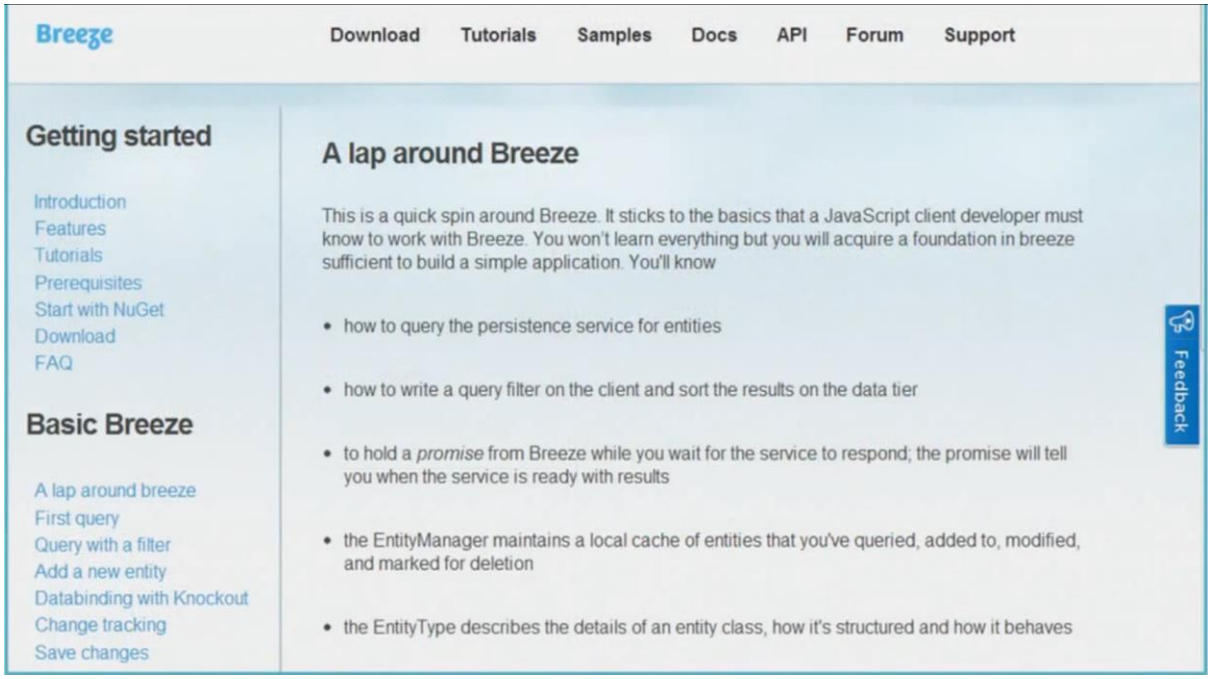


48:05

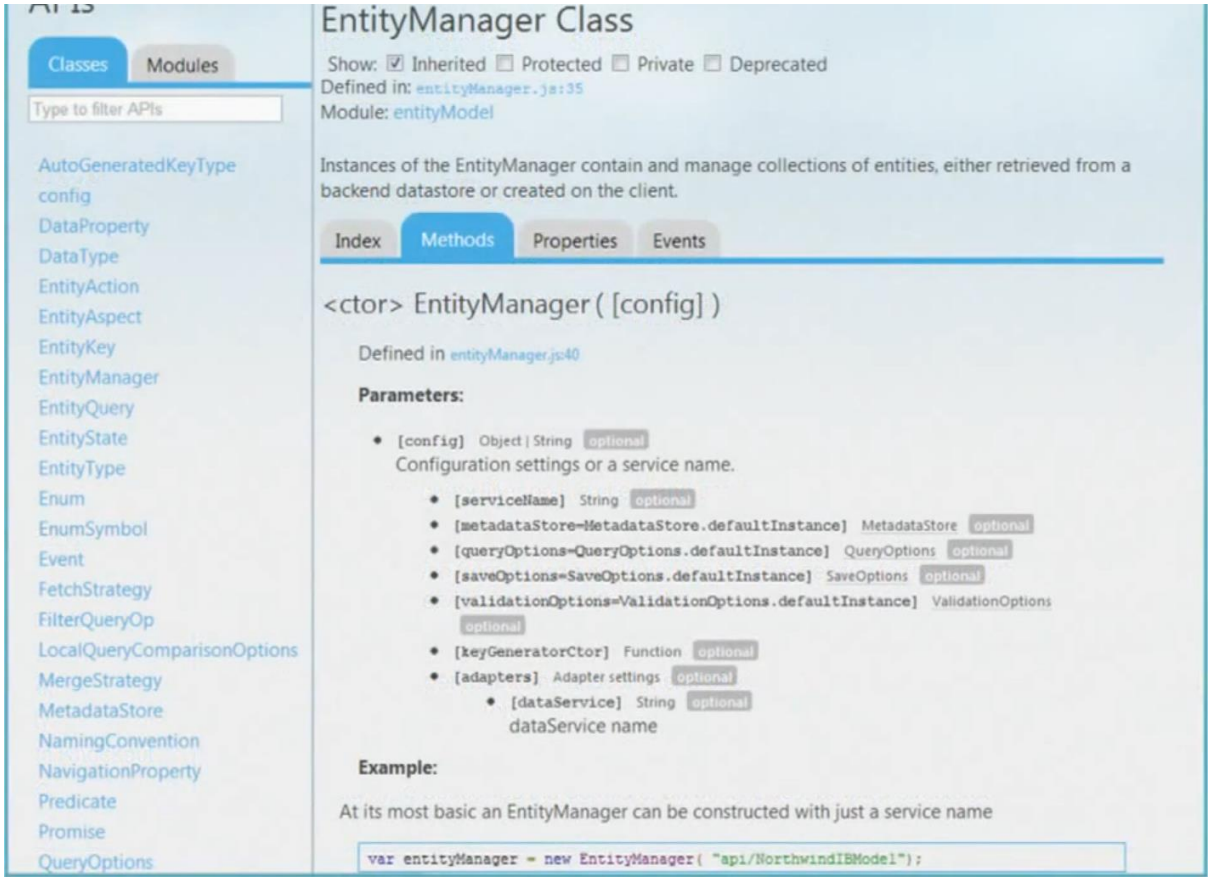
So where can you find out more about this ***Breeze*** thing?

We have a lot of resources on our http://www.breeze.com site . For example there's a live tutorial so you can play with Angular and querying and all kinds of things there right on screen.

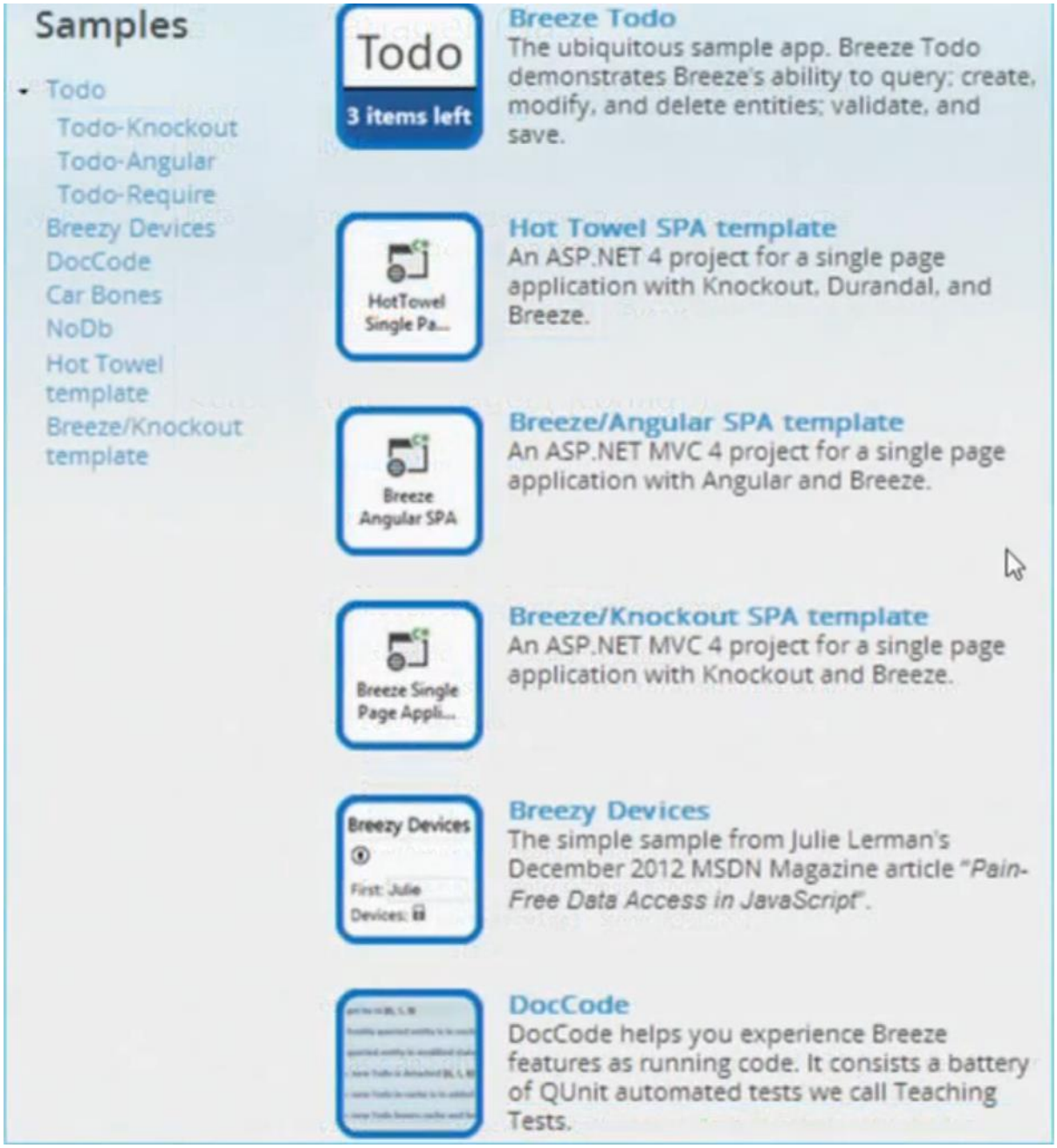




We have lots of documentation that's sort of a guide.



We have API documentation like you would expect for each of the components.



We have samples: a growing set of samples that we have written and we have other people contributing them.

48:45

You can learn more about it at www.breezejs.com and you can follow us on Twitter.

It's 7:53 and I am ready for your questions.

Q&A

Supported Back Ends

Q: I notice that you have all the .NET back end stuff. Do you have the requests that Breeze makes documented such that I could say implement a back-end in Rails or Node or Java or whatever else?

A: Yes. As a matter of fact I have a back-end working against Node. I have an example that I would have loved to have tried it out but I didn't. So that's a pure JavaScript side. We'll be doing a PHP version and a Java version. I don't personally know Rails but I'm hoping to find somebody to do that. But yes, our principle is that you can have any back-end that you want. If it doesn't for example provide metadata, no problem – you can create that on the client-side. If it doesn't have the querying stuff, no problem – you can hit it just any way you want.

I'm making a claim. You've got the samples so that you can see it that we should work with any server - any http server – back end.

REST Client Support

Q: Does it mean that it works with REST clients too?

A: Will it work with a REST client? Again, if it serves HTTP, yes. There are a variety of ways you can do it. There's kind of a hard way the first time but the whole way of interacting with a back-end on the client is there's also this thing called a data service adapter which allows you to morph anything on the way in and the way out. We have an AJAX adapter so you can pull that out and short-circuit it for testing purposes. So yes, there's a variety of ways in which that can be made to happen.

Cyclic Dependency between Entities

Q: How does it handle the cyclic dependency between entities? Say for example a parent has children and the child has a parent. You just use the data model to represent that right, so how does it come to the … basically like NHibernate you say parent has children…

A: All of that would be on the server side. There is the whole language of whether to use NHibernate or any of the other ORMs or something like that. The key question is how do I communicate those relationships across? The only standard we know for representing data of this kind is OData. It doesn't have to be an OData back end but if your server can describe, and NHibernate can, people who have front-ends for it they can, can describe it or if it actually is an OData back end no problem. If it isn't, if you can describe it with OData metadata and we're providing components for you to do so then it's good – you can just send stuff from the server. If not you can describe the metadata and all of the relationships in JavaScript with a MetadataStore object and actually you could cache that and serve that. You could keep it locally so you could make it very fast if you wanted to.

Q: So does it deal with cyclic dependencies?

A: You mean like orders have customers, customers say what's my orders? Yeah, yeah, yeah. It breaks all that. It breaks that chain. It knows all about that. Absolutely. I know what you mean because you can get into serialization trouble and virtually any real model has cycles in it. So, yeah, yeah yeah.

Q: <indecipherable>

A: You're talking server-side stuff now. So the answer depends on the server. In this case I'm using JSON.NET and having it break the cycles but also the way we represent the data, the way we configure it to represent the data breaks cycles.

[Session ends]