Backbone.js Wine Cellar Tutorial — Part 3: Deep Linking and Application States

by **Christophe Coenraets** on December 8, 2011 in **Backbone.js**, **JQuery**, **REST**

UPDATE: I posted a "Postface" to this series with some lessons learned and an improved version of the app. Make sure you read it here.

In Part 1 of this tutorial, we set up the basic infrastructure for the Wine Cellar application. In Part 2, we added the ability to create, update, and delete (CRUD) wines.

There are a few remaining issues in the application. They are all related to "deep linking". The application needs to continuously keep its URL in sync with its current state. This allows you to grab the URL from the address bar at any point in time during the course of the application, and re-use or share it to go back to the exact same state.

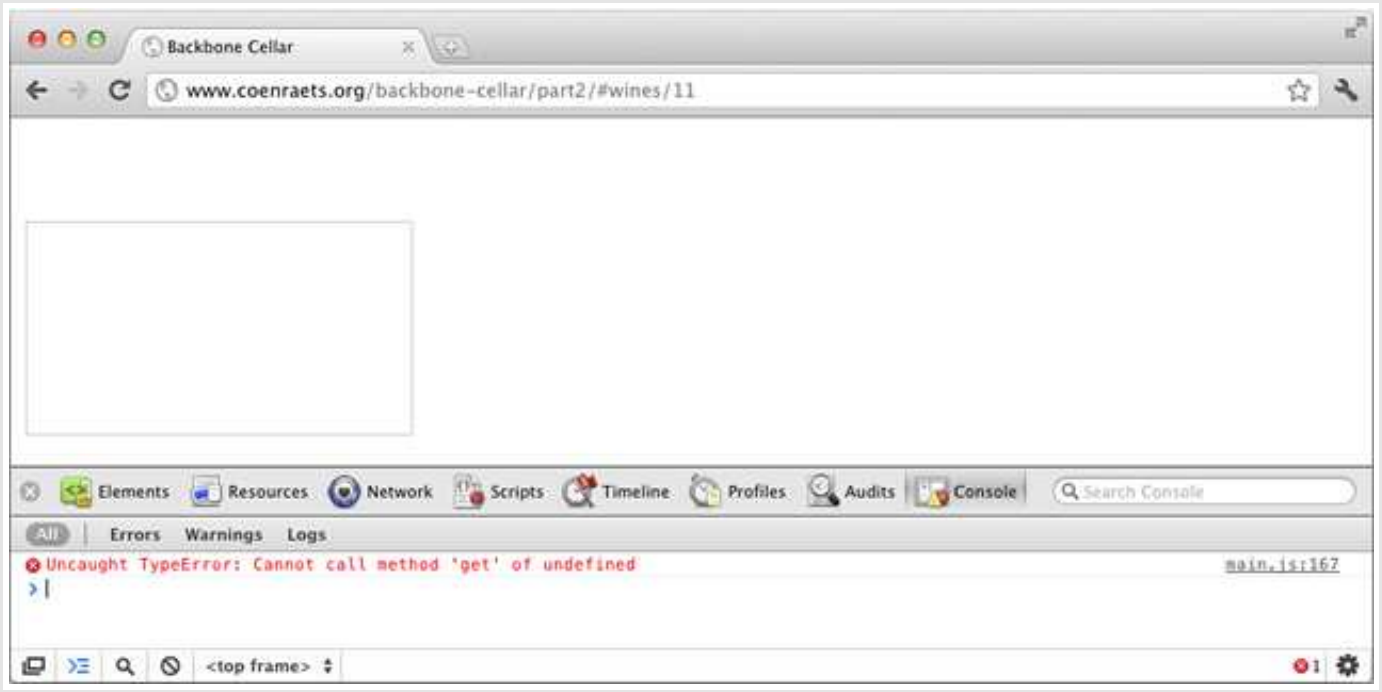
In this last installment of the tutorial, we add support for deep linking in the Wine Cellar application.

Problem 1: Linking to a specific wine

The problem: Select a specific wine from the list. The URL looks like this: http://www.coenraets.org/backbone-cellar/part2/#wines/*[id]*. Now do one of these two things:

1. Grab that URL from the address bar and try to access it from another browser window or tab
2. Simply click your browser's Refresh button

You get an empty screen with no data. If you look at a debug console (for example, in Chrome's Developer Tools), you'll get this message:



Let's take a look at what's going on here:

```
1  var AppRouter = Backbone.Router.extend({
2
3      routes:{
4          "":"list",
5          "wines/:id":"wineDetails"
6      },
7
8      initialize:function () {
9          $('#header').html(new HeaderView().render().el);
10     },
11
12     list:function () {
13         this.wineList = new WineCollection();
14         this.wineListView = new WineListView({model:this.wineList});
15         this.wineList.fetch();
16         $('#sidebar').html(this.wineListView.render().el);
17     },
18
19     wineDetails:function (id) {
20         this.wine = this.wineList.get(id);
21         if (app.wineView) app.wineView.close();
22         this.wineView = new WineView({model:this.wine});
23         $('#content').html(this.wineView.render().el);
24     }
25
26 });
```

The problem is on line 20: we are assuming that a wine collection (this.wineList) already exists, and are trying to “get” a specific item from that list. That works well when we start the application with the default (“”) route. But this.wineList won’t exist if we start the application with the “wines/:id” route. There are different ways to address the issue:

We could modify the wineDetails function to fetch the requested item directly:

```
1  wineDetails: function(id) {
2      this.wine = new Wine({id: id});
3      this.wineView = new WineView({model: this.wine});
4      this.wine.fetch();
5  }
```

That takes care of loading the wine details in the form, but the wine list remains empty when we start the application with “wines/:id” route. We could add the following line of code to load the list if it doesn’t exist:

```
1  if (!this.wineList) this.list();
```

But now, the wine model that is part of the collection and the wine model fetched separately are two different objects, which means that data binding and View synchronization will not work as expected.

Another approach is to check if the collection exists in the wineDetails function. If it does, we simply “get” the requested item and render it as we did before. If it doesn’t, we store the requested id in a variable, and then invoke the existing list() function to populate the list. We then modify the list function: When we get the list from the server (on success), we check if there was a requested id. If there was, we invoke the wineDetails function to render the corresponding item.

```
var AppRouter = Backbone.Router.extend({

    routes:{
        "":"list",
        "wines/new":"newWine",
        "wines/:id":"wineDetails"
    },

    initialize:function () {
        $('#header').html(new HeaderView().render().el);
    },

    list:function () {
        this.wineList = new WineCollection();
        var self = this;
        this.wineList.fetch({
            success:function () {
                self.wineListView = new WineListView({model:self.wineList});
                $('#sidebar').html(self.wineListView.render().el);
                if (self.requestedId) self.wineDetails(self.requestedId);
            }
        });
    },

    wineDetails:function (id) {
        if (this.wineList) {
            this.wine = this.wineList.get(id);
            if (this.wineView) this.wineView.close();
            this.wineView = new WineView({model:this.wine});
            $('#content').html(this.wineView.render().el);
        } else {
            this.requestedId = id;
            this.list();
        }
    },

    newWine:function () {
        if (app.wineView) app.wineView.close();
        app.wineView = new WineView({model:new Wine()});
        $('#content').html(app.wineView.render().el);
    }

});
```

Problem 2: Updating the URL after a wine is created

The problem: Add a new Wine, and click Save. The id that has been assigned to the newly created wine appears in the form field. However the URL is still:
http://localhost/backbone-cellar/part2/ when it should really be: http://localhost/backbone-cellar/part2/#wines/*[id]*.

You can easily fix that issue by using the router's navigate function to change the URL. The second argument (false), indicates that we actually don't want to "execute" that route: we just want to change the URL.

```
if (this.model.isNew()) {
    var self = this;
    app.wineList.create(this.model, {
        success: function() {
            app.navigate('wines/'+self.model.id, false);
        }
    });
} else {
    this.model.save();
}
```

Problem 3: Updating the URL while creating a new wine

The problem: Select a wine in the list. The URL looks like this: http://localhost/backbone-cellar/part2/#wines/*[id]*

Now, click the "Add Wine" button. You get an empty form to enter a new wine, but notice that the URL is still unchanged (http://localhost/backbone-cellar/part2/#wines/*[id]*). In other words, it doesn't reflect the current state of the application.

In this case, we are going to add a new "route":

```
routes: {
    ""          : "list",
    "wines/new" : "newWine",
    "wines/:id" : "wineDetails"
},
```

The router's newWine method is implemented as follows:

```
newWine: function() {
    console.log('MyRouter newWine');
    if (app.wineView) app.wineView.close();
    app.wineView = new WineView({model: new Wine()});
    app.wineView.render();
}
```

And we can change the HeaderView newWine() method as follows:

```
newWine: function(event) {
    app.navigate("wines/new", true);
    return false;
}
```

Putting it all together

You can run the application (Part 3) here. The create/update/delete features are disabled in this online version. Use the link at the bottom of this post to download a fully enabled version.

Here is the final version of the code:

```
// Models
window.Wine = Backbone.Model.extend({
    urlRoot:"../api/wines",
    defaults:{
        "id":null,
        "name":"",
        "grapes":"",
        "country":"USA",
        "region":"California",
        "year":"",
        "description":"",
        "picture":""
    }
});

window.WineCollection = Backbone.Collection.extend({
```

```
    model:Wine,
    url:"../api/wines"
});


// Views
window.WineListView = Backbone.View.extend({

    tagName:'ul',

    initialize:function () {
        this.model.bind("reset", this.render, this);
        var self = this;
        this.model.bind("add", function (wine) {
            $(self.el).append(new WineListItemView({model:wine}).render().el);
        });
    },

    render:function (eventName) {
        _.each(this.model.models, function (wine) {
            $(this.el).append(new WineListItemView({model:wine}).render().el);
        }, this);
        return this;
    }
});

window.WineListItemView = Backbone.View.extend({

    tagName:"li",

    template:_.template($('#tpl-wine-list-item').html()),

    initialize:function () {
        this.model.bind("change", this.render, this);
        this.model.bind("destroy", this.close, this);
    },

    render:function (eventName) {
        $(this.el).html(this.template(this.model.toJSON()));
        return this;
    },

    close:function () {
        $(this.el).unbind();
```

```
61          $(this.el).remove();
62      }
63  });
64
65  window.WineView = Backbone.View.extend({
66
67      template:_.template($('#tpl-wine-details').html()),
68
69      initialize:function () {
70          this.model.bind("change", this.render, this);
71      },
72
73      render:function (eventName) {
74          $(this.el).html(this.template(this.model.toJSON()));
75          return this;
76      },
77
78      events:{
79          "change input":"change",
80          "click .save":"saveWine",
81          "click .delete":"deleteWine"
82      },
83
84      change:function (event) {
85          var target = event.target;
86          console.log('changing ' + target.id + ' from: ' + target.defaultValue + ' to: ' + target.value);
87          // You could change your model on the spot, like this:
88          // var change = {};
89          // change[target.name] = target.value;
90          // this.model.set(change);
91      },
92
93      saveWine:function () {
94          this.model.set({
95              name:$('#name').val(),
96              grapes:$('#grapes').val(),
97              country:$('#country').val(),
98              region:$('#region').val(),
99              year:$('#year').val(),
100             description:$('#description').val()
101         });
102         if (this.model.isNew()) {
103             var self = this;
104             app.wineList.create(this.model, {
```

```
                success:function () {
                    app.navigate('wines/' + self.model.id, false);
                }
            });
        } else {
            this.model.save();
        }

        return false;
    },

    deleteWine:function () {
        this.model.destroy({
            success:function () {
                alert('Wine deleted successfully');
                window.history.back();
            }
        });
        return false;
    },

    close:function () {
        $(this.el).unbind();
        $(this.el).empty();
    }
});

window.HeaderView = Backbone.View.extend({

    template:_.template($('#tpl-header').html()),

    initialize:function () {
        this.render();
    },

    render:function (eventName) {
        $(this.el).html(this.template());
        return this;
    },

    events:{
        "click .new":"newWine"
    },

```

```
    newWine:function (event) {
        app.navigate("wines/new", true);
        return false;
    }
});


// Router
var AppRouter = Backbone.Router.extend({

    routes:{
        "":"list",
        "wines/new":"newWine",
        "wines/:id":"wineDetails"
    },

    initialize:function () {
        $('#header').html(new HeaderView().render().el);
    },

    list:function () {
        this.wineList = new WineCollection();
        var self = this;
        this.wineList.fetch({
            success:function () {
                self.wineListView = new WineListView({model:self.wineList});
                $('#sidebar').html(self.wineListView.render().el);
                if (self.requestedId) self.wineDetails(self.requestedId);
            }
        });
    },

    wineDetails:function (id) {
        if (this.wineList) {
            this.wine = this.wineList.get(id);
            if (this.wineView) this.wineView.close();
            this.wineView = new WineView({model:this.wine});
            $('#content').html(this.wineView.render().el);
        } else {
            this.requestedId = id;
            this.list();
        }
    },

```

```
193         newWine:function () {
194             if (app.wineView) app.wineView.close();
195             app.wineView = new WineView({model:new Wine()});
196             $('#content').html(app.wineView.render().el);
197         }
198
199     });
200
201     var app = new AppRouter();
202     Backbone.history.start();
```

Download

The source code for this application is hosted on GitHub here (see part3). And here is a quick link to the download.

You will need the RESTful services to run this application. A PHP version (using the Slim framework) is available as part of the download.

UPDATE (1/11/2012): A version of this application with a Java back-end (using JAX-RS and Jersey) is also available on GitHub here. You can find more information on the Java version of this application here.

Share this Article:

Subscribe

Related Posts:

Sample iOS Application with Xcode, Objective-C, Storyboard, and Core Data
Keypoint: PhoneGap-Based HTML Slide Decks
The Create Now Tour starts this Thursday in San Francisco
Building Modular Mobile/PhoneGap Apps with Backbone.js, RequireJS & Topcoat — Sample App
Building Modular Web Applications with Backbone.js and RequireJS — Sample App

‹ Backbone.js Wine Cellar Tutorial — Part 2: CRUD

Fact Check Michele Bachmann with Politifact and Flex ›

Search...

About Me

I'm a Technical Evangelist for Adobe where I focus on the Web Platform, Mobile Applications and Enterprise Integration.

Read More

Subscribe

POPULAR LATEST COMMENTS TAGS

PhoneGap and Cordova with iOS 7
SEPTEMBER 19, 2013

Sociogram Mobile: A Starter-Kit Application for PhoneGap and Facebook Integration
MARCH 21, 2013

Architecting a PhoneGap Application: Video + Slides
MAY 13, 2013

Building Modular Mobile/PhoneGap Apps with Backbone.js, RequireJS & Topcoat — Sample App
JUNE 27, 2013

Sample Mobile / PhoneGap Application with Backbone.js and Topcoat
JUNE 4, 2013

Recent Tweets

Follow **@ccoenraets** on Twitter