Backbone.js Wine Cellar Tutorial — Part 2: CRUD

by **Christophe Coenraets** on December 7, 2011 in **Backbone.js**, **REST**

In Part 1 of this tutorial, we set up the basic infrastructure for the Wine Cellar application. The application so far is read-only: it allows you to retrieve a list of wines, and display the details of the wine you select.

In this second installment, we will add the ability to create, update, and delete (CRUD) wines.

RESTful Services

As mentioned in Part 1, Backbone.js provides a natural and elegant integration with RESTful services. If your back-end data is exposed through a pure RESTful API, retrieving (GET), creating (POST), updating (PUT), and deleting (DELETE) models is incredibly easy using the Backbone.js simple Model API.

This tutorial uses pure RESTful services. The services are implemented as follows:

HTTP Method	URL	Action
GET	/api/wines	Retrieve all wines
GET	/api/wines/10	Retrieve wine with id == 10
POST	/api/wines	Add a new wine
PUT	/api/wines/10	Update wine with id == 10
DELETE	/api/wines/10	Delete wine with id == 10

A PHP version of these services (using the Slim framework) is available as part of the download. A similar Java version of the API (using JAX-RS) is available as part of this post.

Using Backbone.js with non-RESTful Services

If your persistence layer is not available through RESTful services, you can override Backbone.sync. From the documentation:

"Backbone.sync is the function that Backbone calls every time it attempts to read or save a model to the server. By default, it uses (jQuery/Zepto).ajax to make a RESTful JSON request. You can override it in order to use a different persistence strategy, such as WebSockets, XML transport, or Local Storage."

Using non-RESTful services is not discussed in this tutorial. See the documentation for more information.

Part 2: Adding Create, Update, Delete

You can run the application (Part 2) 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 code for the improved version of the applications. Key changes are discussed below.

```
1  // Models
2  window.Wine = Backbone.Model.extend({
3      urlRoot:"../api/wines",
4      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();
        $(this.el).remove();
    }
});

window.WineView = Backbone.View.extend({

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

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

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

    events:{
        "change input":"change",
        "click .save":"saveWine",
        "click .delete":"deleteWine"
    },

    change:function (event) {
        var target = event.target;
        console.log('changing ' + target.id + ' from: ' + target.defaultValue + ' to: ' + target.value);
        // You could change your model on the spot, like this:
        // var change = {};
        // change[target.name] = target.value;
        // this.model.set(change);
    },

```

```
    saveWine:function () {
        this.model.set({
            name:$('#name').val(),
            grapes:$('#grapes').val(),
            country:$('#country').val(),
            region:$('#region').val(),
            year:$('#year').val(),
            description:$('#description').val()
        });
        if (this.model.isNew()) {
            app.wineList.create(this.model);
        } 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) {
        if (app.wineView) app.wineView.close();
        app.wineView = new WineView({model:new Wine()});
        $('#content').html(app.wineView.render().el);
        return false;
    }
});


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

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

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

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

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

});

var app = new AppRouter();
```

```
181 Backbone.history.start();
```

Wine

Two attributes were added to the Wine Model:

- **urlRoot**: RESTful service endpoint to retrieve or persist Model data. Note that this attribute is only needed when retrieving/persisting Models that are not part of a Collection. If the Model is part of a Collection, the **url** attribute defined in the Collection is enough for Backbone.js to know how to retrieve, update, or delete data using your RESTful API.
- **defaults**: Default values used when a new instance of the model is created. This attribute is optional. However, it was required in this application for the wine-details template to render an 'empty' wine model object (which happens when adding a new wine).

WineListView

When a new wine is added, you want it to automatically appear in the list. To make that happen, you bind the View to the **add** event of the WineListView model (which is the collection of wines). When that event is fired, a new instance of WineListItemView is created and added to the list.

WineListItemView

When a wine is changed, you want the corresponding WineListItemView to re-render automatically to reflect the change. To make that happen, you bind the View to the **change** event of its model, and execute the render function when the event is fired.

Similarly, when a wine is deleted, you want the list item to be removed automatically. To make that happen, you bind the view to the **destroy** event of its model and execute our custom close function when the event is fired. To **avoid memory leaks and events firing multiple times**, it is important to unbind the event listeners before removing the list item from the DOM.

Note that in either case we don't have the overhead of re-rendering the entire list: we only re-render or remove the list item affected by the change.

WineView

In the spirit of encapsulation, the event handlers for the Save and Delete buttons are defined inside WineView, as opposed to

defining them as free-hanging code blocks outside the “class” definitions. You use the Backbone.js Events syntax which uses jQuery delegate mechanism behind the scenes.

There are always different approaches to update the model based on user input in a form:

- “Real time” approach: you use the change handler to update the model as changes are made in the form. This is in essence bi-directional data binding: the model and the UI controls are always in sync. Using this approach, you can then choose between sending changes to the server in real time (implicit save), or wait until the user clicks a Save button (explicit save). The first option can be chatty and unpractical when there are cross-field validation rules. The second option may require you to undo model changes if the user navigates to another item without clicking Save.
- “Delayed” approach: You wait until the user clicks Save to update the model based on the new values in UI controls, and then send the changes to the server.

This discussion is not specific to Backbone.js and is therefore beyond the scope of this post. For simplicity, I used the delayed approach here. However I still wired the **change** event, and use it to log changes to the console. I found this very useful when debugging the application, and particularly to make sure I had cleaned up my bindings (see close function): I you see the change event firing multiple times, you probably didn’t clean up as appropriate.

HeaderView

Backbone.js Views are typically used to render domain models (as done in WineListView, WineListItemView, and Wine View). But they can also be used to create composite UI components. For example, in this application, we define a Header View (a toolbar) that could be made of different components and that encapsulates its own logic.

Download

The source code for this application is hosted on GitHub here (see part2). 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.

What's Next?

The application so far doesn't support deep-linking. For example, select a wine in the list, grab the URL in the address bar and paste it in another browser window: it doesn't work. In Part 3, we will add complete support for deep linking.

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 1: Getting Started

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

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