Christophe Coenraets

RESTful services with jQuery and Java using JAX-RS and Jersey

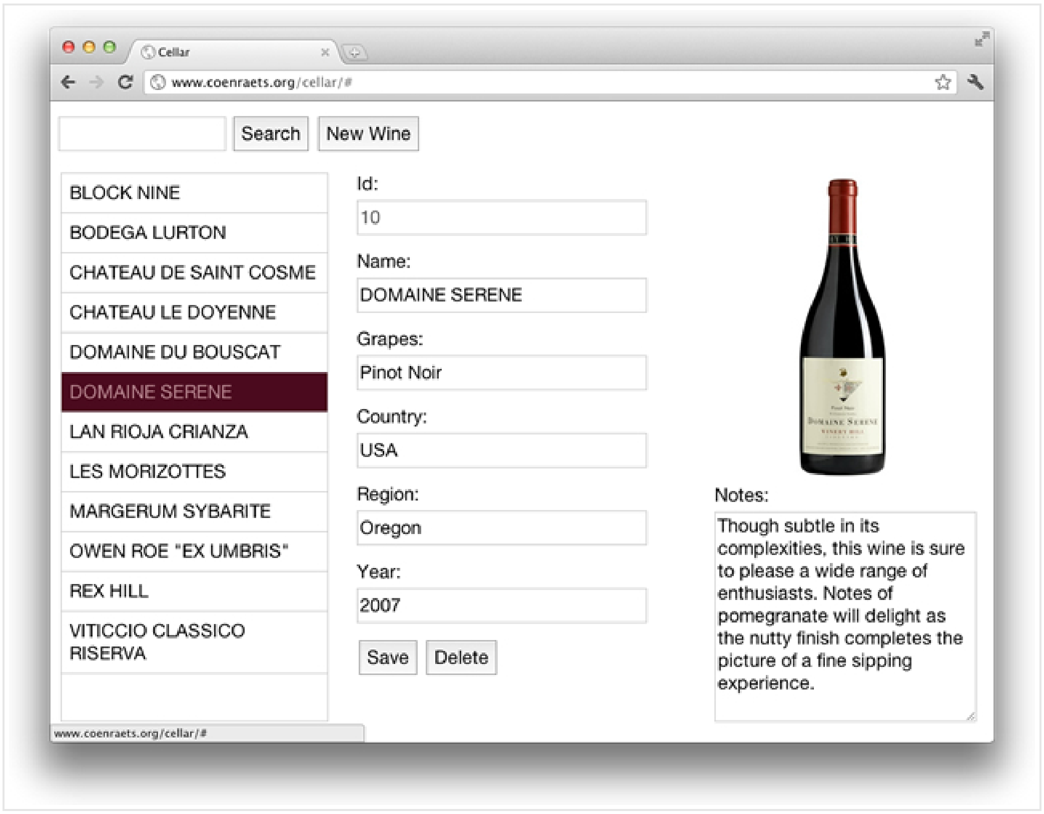
December 1, 2011 in **Java**, **JAX-RS**, **JQuery**, **REST**

NOTE: This is the Java version of this article and its companion app. A PHP version is available here.

This is a more in depth version of my previous post on the same topic. The previous article only covered the HTTP GET method for building RESTful services. This article (and its new companion app) provides an example of building a complete RESTful API using the different HTTP methods:

- **GET** to retrieve and search data
- **POST** to add data
- **PUT** to update data
- **DELETE** to delete data

The application used as an example for this article is a Wine Cellar app. You can search for wines, add a wine to your cellar, update and delete wines.



BLOCK NINE
BODEGA LURTON
CHATEAU DE SAINT COSME
CHATEAU LE DOYENNE
DOMAINE DU BOUSCAT
DOMAINE SERENE
LAN RIOJA CRIANZA
LES MORIZOTTES
MARGERUM SYBARITE
OWEN ROE "EX UMBRIS"
REX HILL
VITICCIO CLASSICO RISERVA

Id: 10
Name: DOMAINE SERENE
Grapes: Pinot Noir
Country: USA
Region: Oregon
Year: 2007
Notes: Though subtle in its complexities, this wine is sure to please a wide range of enthusiasts. Notes of pomegranate will delight as the nutty finish completes the picture of a fine sipping experience.

You can run the application 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.

The REST API consists of the following methods:

Method	**URL**	**Action**
GET	/api/wines	Retrieve all wines
GET	/api/wines/search/Chateau	Search for wines with ‘Chateau’ in their name

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

Implementing the API using JAX-RS

JAX-RS makes it easy to implement this API in Java. You simply create a class defined as follows:

```
package org.coenraets.cellar;

@Path("/wines")
public class WineResource {

 WineDAO dao = new WineDAO();

 @GET
 @Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
 public List<Wine> findAll() {
  return dao.findAll();
 }

 @GET @Path("search/{query}")
 @Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
 public List<Wine> findByName(@PathParam("query") String query) {
  return dao.findByName(query);
 }

 @GET @Path("{id}")
 @Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
 public Wine findById(@PathParam("id") String id) {
  return dao.findById(Integer.parseInt(id));
 }

 @POST
 @Consumes({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
 @Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
 public Wine create(Wine wine) {
  return dao.create(wine);
 }

 @PUT @Path("{id}")
 @Consumes({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
 @Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
 public Wine update(Wine wine) {
  return dao.update(wine);
 }

 @DELETE @Path("{id}")
 @Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
 public void remove(@PathParam("id") int id) {
  dao.remove(id);
  }
}
```
Quick look at the JAX-RS annotations used in this class:

- **@GET**, **@POST**, **@PUT**, **@DELETE**: HTTP method the class method responds to.
- **@Path**: path the method responds to.
- **@Consumes**: type of data the method can take as input. The data will automatically be deserialized into a method input parameter. For example, you can pass a wine object to the addWined() method either as JSON or XML. The JSON or XML representation of a new wine is automatically deserialized into the Wine object passed as an argument to the method.
- **@Produces**: One or more response content type(s) the method can generate. The method's return value will be automatically serialized using the content type requested by the client. If the client didn't request a specific content type, the first content type listed in the @Produces annotation will be used. For example, if you access http://coenraets.org/rest/wines, you get a list of wines represented as JSON because it is the first content type listed in the @Produces annotation of the findAll() method.

The jQuery client below sends data to the server using JSON (addWine() and updateWine() methods).

The approach you use to actually retrieve the data is totally up to you. In this example, I use a simple DAO, but you can of course use your own data access solution.

Testing the API using cURL

If you want to test your API before using it in a client application, you can invoke your REST services straight from a browser address bar. For example, you could try:

- http://localhost:8080/cellar/rest/wines
- http://localhost:8080/cellar/rest/wines/search/Chateau
- http://localhost:8080/cellar/rest/wines/5

You will only be able to test your GET services that way, and even then, it doesn't give you full control to test all the content types your API can return.

A more versatile solution to test RESTful services is to use cURL, a command line utility for transferring data with URL syntax.

For example, using cURL, you can test the Wine Cellar API with the following commands:

- Get all wines returned as default content type:
  `curl -i -X GET http://localhost:8080/cellar/rest/wines`
- Get all wines returned as xml:
  `curl -i -X GET http://localhost:8080/cellar/rest/wines -H 'Accept:application/xml'`
- Get all wines with 'chateau' in their name:
  `curl -i -X GET http://localhost:8080/cellar/rest/wines/search/chateau`
- Get wine #5:
  `curl -i -X GET http://localhost:8080/cellar/rest/wines/5`
- Delete wine #5:
  `curl -i -X DELETE http://localhost:8080/cellar/rest/wines/5`
- Add a new wine:
  `curl -i -X POST -H 'Content-Type: application/json' -d '{"name": "New Wine", "year": "2009"}' http://localhost:8080/cellar/rest/wines`
- Modify wine #27:
  `curl -i -X PUT -H 'Content-Type: application/json' -d '{"id": "27", "name": "New Wine", "year": "2010"}' http://localhost:8080/cellar/rest/wines/27`

The jQuery Client

Accessing your API through cURL is cool, but there is nothing like a real application to put your API to the test. So the source code (available for download at the end of this post) includes a simple jQuery client to manage your wine cellar.

Here is the jQuery code involved in calling the services:

```
function findAll() {
 $.ajax({
  type: 'GET',
  url: rootURL,
  dataType: "json", // data type of response
  success: renderList
 });
}

function findByName(searchKey) {
 $.ajax({
  type: 'GET',
  url: rootURL + '/search/' + searchKey,
  dataType: "json",
  success: renderList
 });
}

function findById(id) {
 $.ajax({
  type: 'GET',
  url: rootURL + '/' + id,
  dataType: "json",
  success: function(data){
   $('#btnDelete').show();
   renderDetails(data);
  }
 });
}

function addWine() {
 console.log('addWine');
 $.ajax({
  type: 'POST',
  contentType: 'application/json',
  url: rootURL,
  dataType: "json",
  data: formToJSON(),
  success: function(data, textStatus, jqXHR){
   alert('Wine created successfully');
   $('#btnDelete').show();
   $('#wineId').val(data.id);
  },
  error: function(jqXHR, textStatus, errorThrown){
   alert('addWine error: ' + textStatus);
  }
 });
}

function updateWine() {
 $.ajax({
  type: 'PUT',
  contentType: 'application/json',
  url: rootURL + '/' + $('#wineId').val(),
  dataType: "json",
  data: formToJSON(),
  success: function(data, textStatus, jqXHR){
   alert('Wine updated successfully');
  },
  error: function(jqXHR, textStatus, errorThrown){
   alert('updateWine error: ' + textStatus);
  }
 });
}

function deleteWine() {
 console.log('deleteWine');
 $.ajax({
  type: 'DELETE',
  url: rootURL + '/' + $('#wineId').val(),
  success: function(data, textStatus, jqXHR){
   alert('Wine deleted successfully');
  },
  error: function(jqXHR, textStatus, errorThrown){
   alert('deleteWine error');
  }
 });
}

// Helper function to serialize all the form fields into a JSON string
function formToJSON() {
 return JSON.stringify({
  "id": $('#id').val(),
  "name": $('#name').val(),
  "grapes": $('#grapes').val(),
  "country": $('#country').val(),
  "region": $('#region').val(),
  "year": $('#year').val(),
  "description": $('#description').val()
  });
}
```

Download the Source Code

The source code for this application is hosted on GitHub here. And here is a quick link to the project download (Eclipse Dynamic Web Project). It includes both the Java and jQuery code for the application.

UPDATE (1/11/2012): A version of this application using Backbone.js at the client-side is also available on GitHub here. You can find more information on the Backbone.js of this application here.

I'm interested in your feedback. Let me know what you think and what your experience has been building RESTful-based applications using Java and jQuery.

Tweet   Follow ccoenraets

Follow Me

‹ Set Up an Amazon EC2 Instance with Tomcat and MySQL – 5 Minute Tutorial

RESTful services with jQuery, PHP and the Slim Framework ›

Developer・Speaker・Writer
Developer Evangelist at Salesforce
Full Bio

Follow ccoenraets

Search...

Tutorials

- ECMAScript 6
- Cordova / PhoneGap
- Ionic Framework
- Salesforce Development
- Salesforce1 App

Projects

- OpenFB
- PageSlider
- Ionic / Salesforce Starter
- ForceJS
- ForceServer

Follow Me