Courslets, a golf improvement web service

Peter Battaglia

Discussion

- Project Overview
- Design and Technologies Utilized
  - Rails and REST
    - URLs, URLs, URLs…
  - Rails and Web Services
    - What's exposed as a service?
    - Active Resource
      - What does it do?
      - How do you use it?
  - Web 2.0 "Features"
    - Prototype and Scriptaculous
- Issues Encountered
- Conclusions
- Demo's
  - Site Demo
  - Sample Client Demo

Project Overview

- Purpose
  - To create a REST-based web service and sample client to track golf score statistics and provide custom printable PDF course booklets
- Core Features
  - REST-style architecture
  - Customizable Drag N' Drop Stats Page
  - Auto-Complete Golf Course Search
  - HTTP Authentication for web service clients

Design

- Rails and REST
  - URLs
    - Scaffolding
      - Rails dynamically creates a set of named RESTful routes for accessing your resources via URLs, this is accomplished by adding “map.resources :resource” in the routes.rb
      - creates all the basic CRUD support code for building your application including generic views, an empty model class and stubbed controller methods
    - The URLs map directly to the actions (methods) in your controller
    - Example (rake routes):
      - GET /golfers {:controller=>"golfers", :action=>"index"}
      - POST /golfers {:controller=>"golfers", :action=>"create"}

Design

- Rails and REST
  - Named Routes
    - For accessing the URLs in your Rails app, you can use named routes, generated by Rails
    - Example (rake routes):
      - golf_courses GET /golf_courses {:controller=>"golf_courses", :action=>"index"}
      - golf_courses_url can be used to reference the action shown above
      - <%= link_to "Golf Course List", golf_courses_path -%>

Design

- Rails and REST
  - Custom Routes
    - Because sometimes you don't want to follow conventions…
    - If you need to create custom actions
      - define the method in the controller
      - create the view
      - add the custom routes into our routes.rb
    - Example:
      - map.resources :resource,:collection => {:search => :get}
      - In this example, we create a custom search action to return a collection of resources for us
      - When the URL /search is invoked with HTTP GET, our collection of resources will be returned to us
      - This also creates the URL: search_resources_path

Design

- Rails and REST
  - Custom URLs
    - If you have actions that you would like to map to “clean” URLs, you can do this…
    - map.login '/login', :controller => 'session', :action => 'new'

Design

- Rails and Web Services
  - Rails has web service support baked right in!
  - Example:

```
def index
   @golfers = Golfer.find(:all, :order => "first_name ASC")
    respond_to do |format|
              format.html # index.html.erb
              format.xml  { render :xml => @golfers }
    end
 end
```

  - In this example, we can either render HTML for our standard client, or if the client is requesting XML, we return them our list of golfers in XML format
  - You can also create custom formats to send back to the client (e.g. respond to iPhone/iPod users)
  - Rails also provides URLs for your web service clients:
    - formatted_golfers GET   /golfers.:format {:controller=>"golfers", :action=>"index"}

Design

- Rails and Web Services
  - Active Resource Clients
    - Active Resource connects business objects and REST web services.
    - AR implements object-relational mapping for REST web services to provide transparent proxying capabilities between a client (ActiveResource) and a RESTful service
    - Model classes are mapped to remote REST resources by Active Resource much the same way Active Record maps model classes to database tables
    - When a request is made to a remote resource, a REST XML request is generated, transmitted, and the result received and serialized into a usable Ruby object
    - AR is built on a standard XML format for requesting and submitting resources over HTTP

Design

- Rails and Web Services
  - Building an Active Resource Client
    - To build a sample client
      - generate a rails application, and controller
      - Create a new class that extends ActiveResource

```
class Golfer < ActiveResource::Base
   self.site = http://pbatt:12345@pclnxpbattaglia:3000/
end
```

    - Then access the resource as you would in ActiveRecord

```
@golfers = Golfers.find(:all)
```

Design

- Rails and Web 2.0
  - Rails "Helpers"
    - Rails provides helper methods for utilizing the Prototype and Scriptaculous Libraries for creating AJAX controls and interesting visual effects
    - Examples:
      - draggable_element("my_image", :revert => true)
      - sortable_element("my_list", :url => { :action => "order" })
      - link_to_remote "Reload", :update => "posts", :url => { :action => "reload" }, :complete => visual_effect(:highlight, "posts", :duration => 0.5)
    - Why Not Use them??
      - Too much magic for me…

Design

- Rails and Web 2.0
  - The manual way…
    - Define CSS, define layout, using divs, and add JavaScript…

```
<script>
  document.observe('dom:loaded', function() {
  var options = {
            hoverclass: 'hover',
            constraint: false, containment: ['left', 'right'],
            dropOnEmpty: true, onUpdate: function(list) {
            var methodStart = list.down('li') ? 'remove' : 'add';
            list[methodStart + 'ClassName']('empty');
  }};
            Sortable.create('left', options);
            Sortable.create('right', options);
  });
</script>
```

Issues

- No real issues…
- Rails has a tendency to hide the details from the developer…fixing issues can take a long time.
- Styling…hacking CSS can be rough.
- Creating a site that’s compatible on all platforms and browser sizes is a real pain.

Conclusions

- Rails is a phenomenal framework that promotes productivity
- Building Web Services, and Web Service clients on Rails is simple and often enjoyable

Demo’s

- Site Demo
- Sample Client