WEB 2.0 AND COMMUNICATION IN NONPROFIT ORGANIZATIONS:

A Case Study and Proof of Concept.

William Clerico
clericow@bc.edu

Boston College
Computer Science Department
Senior Thesis

Advisor: William Ames
ameswi@bc.edu

TABLE OF CONTENTS

FORWARD

Dedication

The list of people to thank for their help with this thesis is far too long to be all encompassed here, but I will attempt to mention a few.

Professor Ames - Your bravery in agreeing to advise a thesis on a technology which neither of us knew anything about - as well as your regular input and technical support - were invaluable. Thanks for keeping me on task while still being flexible and understanding.

Professor Gallaugher - The insights I gained into technology strategy in your courses shaped the way I approached this thesis, and helped me analyze the exact needs of the organization.

Hugh Drummond - Your insight into the American Red Cross's use of the Internet was some of the most realistic advice that I received throughout this whole process and provided a great foundation for the rest of my research.

John Solman - The desire that you express to improve the operations of the Red Cross and the time you commit to doing so are inspiring. Thanks for your advice, time, and effort.

Mike Klau & the Monday Night Team - Mike, the passion and regularity with which you serve others is inspiring, and your charismatic style of leadership is unrivaled. My four years on your team have shaped the way I view volunteering and non-profits in general - not merely as a result of your leadership, but also through my interactions with the team members.

Mom & Dad - Thanks for putting a roof over my head and taking care of me when I had to come home to meet deadlines. Your support and love is what sustains me.

Jen Kaye & Becca LaPlante - The late night entertainment and motivation that you both provided in the study lounge is a large part of the reason why I'm not still writing this thesis right now. Sorry about all the swearing.

Jeff, Tyler, Nick, & Bagley - Though I would have had much more time to work on this without your distractions, I can't help but say thanks for how much fun you made my senior year, and how sane you kept me during my all nighters.

Why this topic?

In high school, volunteerism was something that everyone did; a necessary part of getting into a good college. Once in college, however, it took on a much more elective role. I was fortunate enough to give it a try, and ended up becoming a regular volunteer with the Disaster Services department of the American Red Cross of Massachusetts Bay.

Coordinating a team of volunteers is a difficult challenge in any activity, but especially so in disaster services when time and information are of the essence. As I took on more responsibility, eventually becoming a team leader, I became frustrated by various problems involving communication and record keeping - problems that I felt could be easily solved through the application of technology.

This thesis is by no means a complete application or complete exploration of the possibilities, it is merely an attempt to address some of the issues and show how technology can be used to solve them.

CHAPTER 1: THE OPPORTUNITY

The Non Profit Sector's $100 Billion Opportunity

Sector Introduction

The nonprofit sector is a large piece of the U.S. economy with tremendous growth. There are 1.5 million different organizations with revenues of $670 billion annually. 109 million Americans volunteer each year.[1] With such large amounts of money flowing through these organizations each year, there has been several studies looking at the efficiencies with which they operate.

In 2003, the *Harvard Business Review* published a comprehensive study of the nonprofit sector, specifically the largest 200,000 organizations which have revenue of more than $25,000 every year.[2] They identified five major areas for improvement:

1. Reduce funding costs
2. Distribute holdings faster
3. Reduce program service costs
4. Trim administrative costs
5. Improve sector effectiveness

The authors of the article made several suggestions about how to improve these various areas, even brushing upon the topic of technology by way of online solicitation donation. However, they also mentioned the difficulties of improving efficiency without significant time and capital investment. It is also difficult to talk in the general sense about the application of technology when the scope of nonprofit organizations is so broad and so varied.

The American Red Cross

The American Red Cross is a national organization that offers services in six areas: domestic disaster relief; community services that help the needy; support and comfort to military members and their families; the collection, processing and distribution of lifesaving blood and blood products; educational programs that promote health and safety; and inter-

[1] "What You Should Know About Nonprofits". National Center for Nonprofit Boards & Independent Sector.

[2] Bill Bradley, Paul Jansen, and Les Silverman. "The NonProfit Sector's $100 Billion Opportunity". *The Harvard Business Review,* May 2003.

national relief and development programs.[3] Nationally, they have over one million volunteers and 35,000 employees spread across 800 chapters. They assist the victims of over 70,000 disasters every year, and pride themselves upon the fact that 91 cents out of every dollar they spend is spent directly on services and programs. They are completely donor funded.

The case study presented here focuses exclusively on the Massachusetts Bay chapter, the seventh largest chapter in the nation.[4] The information gathered comes from personal observation while working within the chapter, or from interviews conducted with employees and volunteers.

As previously mentioned, the American Red Cross provides six difference categories of service. Since the vast majority of my experience is within the Disaster Services department, that is what I focused my research upon, though some of the people interviewed do work across all six service areas. The Disaster Services department in the Massachusetts Bay chapter is one of the chapter's busiest departments, and is also its most fragmented. In 2006, the department responded to 396 local disasters, assisting 2,324 people, and providing $400,000 in financial assistance. In addition, the department educated 61,759 people through community outreach. The department responds primarily to house fires, providing emergency money, food, clothing, and shelter to those displaced by disaster. The department is on call 24 hours a day, 7 days a week, 365 days a year, and averages more than one response per day.

I interviewed Hugh Drummond, director of external relations, about their website (http://www.bostonredcross.org). A company that subsequently went bankrupt developed the site in 2001 pro bono. The site has minimal interactivity, and consists of standard HTML and Javascript. Hugh listed the purposes of the site:

- Communicate information
    - Internally
    - Externally
        - Media
        - Emergency partners
        - Public
- Recruit Volunteers

---

[3] "About Us". American Red Cross. http://www.redcross.org/aboutus

[4] "Monday Night Team". *American Red Cross.* http://www.arcmb.org

- Solicit Donations
- Enroll students in classes
- Allow products to be purchased

Ideally, Hugh said, the site would be customizable by audience and speed collection and dissemination of information in manner which could be easily updated. He pointed to other chapters of the Red Cross which had effective websites, and also pointed to Presidential campaigns as effective users of the web medium to accomplish fundraising and communication goals.

The site is also the victim of a structural challenge at the Red Cross. Since Hugh is the sole person in the public relations department, it is tough for him to dedicate enough time to the site to keep it fresh, especially if technical aptitude is required. The majority of nonprofits simply do not have the manpower or financial resources to appoint full-time webmasters to maintain their web presence.

Another weakness that I observed in the Massachusetts Bay chapter's web strategy was the lack of extensibility. Several volunteers had built various sites separate from the main chapter's site to log hours and keep track of team schedules, but these sites did not integrate with the chapter's main web presence. This lack of oversight worried many, as the reputation of the Red Cross is one of its greatest assets, and as scandals have shown, very vulnerable. Also, some teams had built sites to house team photos and stories. This information would be much more valuable as a donation solicitation tool if it were on the site that donors visited.

While my interview with Hugh focused on his current use of the web, my interviews with John and Mike, both field workers, focused on what tools would help them do their jobs more effectively. Both of them shared their techniques for record keeping, which consisted of a mixture between Microsoft Excel and paper records. When asked what sort of functionality would be most useful, they provided the following answers:

1. Searchable team rosters
2. Simplified contact information
3. Mapping & direction capability
4. Automatic paging & notification
5. Scheduling

John, a recent appointee to the position of North Zone manager, had scheduled appointments with every team leader (of which there are more than ten) in order to gather the most updated rosters. For a volunteer who also works full-time as a pediatric nurse, this is a tremendous expense of effort and time, not to mention the additional labor of consolidating this information and keeping it updated. Also, as the information is passed along and manipulated, the chances for error are compounded. These factors all combine to result in a final product which is outdated, labor-intensive, and unreliable. I immediately identified this dilemma as one easily solved by web technology.

Being a team leader who hails from New Jersey, I constantly found myself unable to visualize the 127 towns and cities that the Massachusetts Bay chapter serves. It was not as simple as plotting the route from my location to the disaster I was responding to - there were many factors which had to be considered, including the location of disaster vehicles and equipment, the location of other volunteers, my location, and the location of the disaster. Typically, when I received a call, it took me 10-15 minutes to plan what I was going to do before I began calling people. This was due to my lack of familiarity with the geography of eastern Massachusetts and directly impacted my team's response time to a disaster, which is a metric measured and recorded by the American Red Cross. Prolonged response time leads to lengthier responses, which can result in the need for a second team to be on call and respond in the event of a second emergency. Supporting and equipping a second team requires further expense and effort, and is redundant if the first team can quickly and nimbly respond.

CHAPTER 2: WEB 2.0

The potential offered by the new web

An Internet platform

During our interview, Hugh indicated the lack of an adequate IT department as one of the major factors behind the Massachusetts Bay chapter's antiquated use of technology. There simply wasn't enough manpower and money to maintain complicated applications or to train staff and volunteers on how to use them. Any application, he said, would have to be simple enough not to require training, and be hosted in such a way that required minimal maintenance by the chapter's staff.

These needs are easily met by the use of the Internet as a platform. Web-based applications, or "software as a service" have grown tremendously popular in the presence of inexpensive bandwidth and pervasive Internet connectivity. In fact, it is projected that IT spending on software as a service (SaaS) will reach $10.7 billion in 2009.[5] The model here is simple - instead of purchasing applications and installing them locally on desktops or servers, companies or organizations pay a monthly or yearly fee for access to a web site. In fact, there are many SaaS tools which already exist for nonprofit organizations as plugins to the popular Salesforce.com software. However, extending this software requires added complexity, and the cost for access is upwards of $1000 for only five licenses.[6] This model is simply too cost prohibitive for an organization like the American Red Cross, which enlists the help of thousands of employees and volunteers.

One drawback on the use of the Internet as a platform is that access to the software can be completely cutoff in the event that data connectivity is compromised. This must be a serious consideration for all organizations, especially those that provide disaster and emergency services. This risk can be mitigated by the appropriate use of power & network backup equipment.

Applications built around data

The 20th century approach to efficient record keeping was a well-designed Microsoft Excel spreadsheet. But in the age of the web, much more information must be collaborated, and the expectation is that it is done with higher accuracy and speed. The practice of emailing spreadsheets is simply obsolete and inefficient.

---

[5] Eric Knoor, Leon Erlanger, and James R. Borck. "A field guide to software as a service". *InfoWorld.* 4/18/05.

[6] "CRM Software Leader". *Salesforce.com.* http://www.salesforce.com

The concept of Web 2.0 is founded upon the idea that applications should not only be web-based, but also share a common copy of data which is shared and manipulated by various clients. This model is a good fit for the fragmented nature of Disaster Services, where many users in many different locations need to access a common, updated copy of information. The code written by the programmer is merely an intermediary that controls presentation and regulates integrity. It sits between the end user and the information that he or she seeks.

Harnessing collective intelligence

The advantage of such data-centric applications is that the information that can be gathered from them is far more accurate and far-reaching than what could be gathered from a highly distributed application. With more data in one place, it is easier to produce correlated reports and to provide greater visibility into the wisdom that the organization already has.

An example of a process at the American Red Cross of Massachusetts Bay which could stand to benefit from harnessing collective intelligence is hours reporting. The Red Cross receives a large amount of its funding from the United Way, which pays the Red Cross a flat amount for each hour that a volunteer works. In order to receive this funding, the Red Cross must document all of the hours that volunteers spend working. There are processes which collect this from paperwork filed after incidents, but there are also a large amount of "non on the scene" hours which must get reported. These hours include attendance at meetings, administrative work, and planning. Currently, these are gathered by way of a monthly email which goes out to all volunteers, and a staff member is tasked with gathering the responses and reporting them properly.

While this task is critical to the funding of the Red Cross, it is also a waste of the staff's time. If there was a web-based medium through which to report hours, these reports could be consolidated and created automatically, freeing the staff up to do less mechanical tasks. In addition to being less time consuming, the quality of the report would also be enhanced, with less chance for error or omission.

CHAPTER 3: THE SHORTCOMINGS OF THE WEB

PHP, JSP, ASP, and the difficulties they present.

PHP

During the initial phases of my research, I was searching for various tools with which to build the web application. Being as it is very popular and versatile, PHP was one of the first technologies I worked with. I built a simple sample application to test its capabilities.

I created an HTML form that fed comments from and information about a user into a database using PHP. The basic architecture was an html form, with a PHP form handler. The form handler established a connection to the database (csweblab.bc.edu) – wrote the data to a table ("feedback") and then closed the connection using SQL.

SQL statement to create table:

CREATE TABLE feedback (comment_id MEDIUMINT UNSIGNED NOT NULL AUTO_INCREMENT, name VARCHAR(20) NOT NULL, email VARCHAR(50), gender varchar (1) NOT NULL, role varchar(1), comments varchar(500), PRIMARY KEY (comment_id))

Issues encountered:

- Tedious data checking is required to protect the integrity of the database
- Tedious technical troubleshooting
- Long development timeline
- Not easily scalable

Advantages:

- Popular & well known language
- Requires no special software besides a web server
- Easily modifiable

I encountered a multitude of technical issues when I was writing in PHP since I was designing the application from scratch. It was very easy to have errors in my SQL queries, and I realize that users could easily wreck my database by improperly answering the form. In production, I would have to use javascript or PHP to do more extensive error checking. While I listed one of the advantages of PHP as not requiring special software, I also en-

countered errors with a difference in PHP versions, resulting in outdated password hashes and other difficulties when connecting to the database.

On the whole, PHP is a very powerful low-level web programming language. However, due to the time constraints of IT departments in the nonprofit sector, I decided that it would not be the best choice for development. Since the functionality that was required by my application design did not require any great degree of complexity, just high level data organization, I decided upon another technology.

Active Server Pages

Active Server Pages, or ASP, is Microsoft's engine for server side scripting.[7] It is highly object oriented, and provides a variety of methods and libraries for creating dynamic web pages, including the use of technologies like Asynchronous Javascript And XML (AJAX), which is an important part of creating responsive web applications.

However, ASP.NET is designed to be run on Microsoft Windows Server, and is obviously a commercial tool. There are open source products like InstantASP and ChiliASP which can run ASP on Linux or other open source operating systems, but these tools must be constantly updated, and can be difficult for an unsophisticated IT department to maintain.[8] ASP also relies heavily on cookies, which can cause security problems and browser incompatibility issues.

Java Server Pages

Java Server Pages, or JSP, is Sun Microsystem's solution for server side scripting. The libraries are closed source, but it runs on free, open source software and relies upon the mechanics of the Java programming language, which is wildly popular. It is also platform independent, meaning that it can be moved amongst multiple servers and machines and adapted to whatever hardware is available. This is highly advantageous to organizations with limited resources.

However, developing websites in JSP is very labor intensive and very complex. There is no simple way to manage objects in relation to the database, or at least not as simple a method as other technologies. Since the high level abstraction of data is one of the central tenets of the application, this is a rather large stumbling block.

---

[7] "The Official Microsoft ASP.NET 2.0 Site". *Microsoft Corporation.* http://www.asp.net

[8] "Introduction to Active Server Pages". *AHref.com.* http://www.ahref.com/guides/technology/199806/0601buz.html

Furthermore, JSP does not run on a standard web server. Instead, it must be run and configured with specialized plugins. Since JSP is actually an extension of the Java servlet API, it is released as part of the Java Enterprise Edition specification. This adds an additional layer of complexity that novice Java developers may not be familiar with. It is also hardware-intensive in that it requires a powerful server with lots of memory to work efficiently.

CHAPTER 4: RUBY ON RAILS

A Web Application Framework

In the end, I decided upon using Ruby on Rails (also known as RoR or Rails) to develop my application. This was partially motivated by selfish reasons: Ruby on Rails developers are in high demand in the current Web 2.0 craze. However, there are strong reasons for its popularity. RoR provides simple abstraction from the database, a host of tools to simplify development, easy interoperability, and elegant object orientation.

Ruby on Rails is open source, and enforces strict adherence to the Model-View-Controller design pattern.

Principles

The first tag-line of Ruby on Rails is "Don't Repeat Yourself" (sometimes shortened to DRY). The idea behind this philosophy is that information in an application is located in one place, where it can be accessed by all parts of the application that require that information.

The second guiding principle of Ruby on Rails is "Convention over Configuration". This means that Ruby is "smart" enough to understand semantics in code, and provides a variety of automatically generated methods and objects to help, all without being asked to by the programmer. For example, Rails is smart enough to know that a table in the database called "profiles" will correspond with a class "profile". Instead of coding all the methods which go along with retrieving fields from a database, packaging them into objects, and relating these objects, one can simple use the methods and relationships which Rails automatically provides.

In the spirit of convention over configuration, Rails also provides functionality known as "scaffolding". Scaffolding is a series of scripts included with the framework that will create various default methods and views based solely upon the variables in a class. For example, creating scaffolding on a sale object might create an rHTML view of the sale object which displayed its price, description, quantity, and customer. It would also automatically generate a view which provided the functionality to change these values.

Model - View - Controller

Model - View - Controller (MVC) is a popular architecture for software design, and the architecture around which Rails is designed. The idea behind MVC is that by separat-

ing an application into these three layers, applications can be easily designed and modified. Changes in one layer do not require sweeping changes in the other to keep the application functional.

ActiveRecord

ActiveRecord is the pattern by which Rails interfaces with the database. In fact, throughout my entire programming experience with RoR, I never once had to write a SQL query. ActiveRecord converts the Ruby classes that the programmer writes into SQL queries, and manages the database directly. One merely calls class methods like "save" and "update_attributes" in order to generate this SQL in the background. In addition to the simplification of code, this provides security benefits as well.

Security

One of the major drawbacks of web-based applications is that they are physically available for all to see and to exploit. An inexperienced web programmer can easily create a site which leaves an entire network vulnerable to hacking, or exposes personal information to the public. Therefore, while this project is not focused on security, it is still important to be conscious of security when designing web applications.

The most basic form of security on a web application is the authentication scheme. Ruby on Rails makes this very simple; in my application I was able to implement SHA1 authentication with only a few lines of code. SHA1, while not completely unbreakable, is widely accepted in the web community as a security standard.[9] With only one additional line of code, I locked down every method on every controller, requiring authentication before allowing the method to be called.

There are a variety of attacks which web applications are vulnerable to: Cross-Site Scripting, SQL injection, session forgery, buffer overflow, etc. However, a major advantage of using Ruby on Rails, as opposed to PHP or other web languages, is the degree of insulation that the framework provides the developer from the database and web server. The methods and practices that are contained within RoR have been examined by the open source community, and are relatively safe and secure.

---

[9] Schneir, Bruce. "Cryptanalysis of SHA-1". http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html

CHAPTER 5: THE APPLICATION

A proof of concept for volunteers

Login

In this application, the first page a potential user is greeted with is the login view of the login controller. This page interrogates the user for a username and password, and provides them the capability to create an account if they do not have one. After authenticating their username and password using the SHA-1 encryption scheme, this controller stores the user's session ID for later use. This will be used to retrieve his or her profile by the portal controller. The Login controller also provides views to create profiles, and methods to control login and logout.

Portal

The central piece of my application is the Portal Controller. This class manages most of the interaction between the user and the database through its many views. It has four main features:

1. Edit Profile
2. Search
3. Availability
4. Map

These features are accessed by way of a layout-based menu. Any view or method included within the portal controller is rendered by way of the portal menu. This conforms with the Don't Repeat Yourself Principle, in that all menu and layout design is done in one place. See the layout code appendix portal.rhtml.

Screenshot of the index view displayed in portal layout.

The index view will prompt the user to create a profile if that user does not have an existing one. If not, it will simply display the user's first name and availability status. This could eventually be expanded to have messaging and alert functions.

The availability method searches all existing profiles and returns a list of profiles which indicate they are available along with the profile's contact information.

Availability View

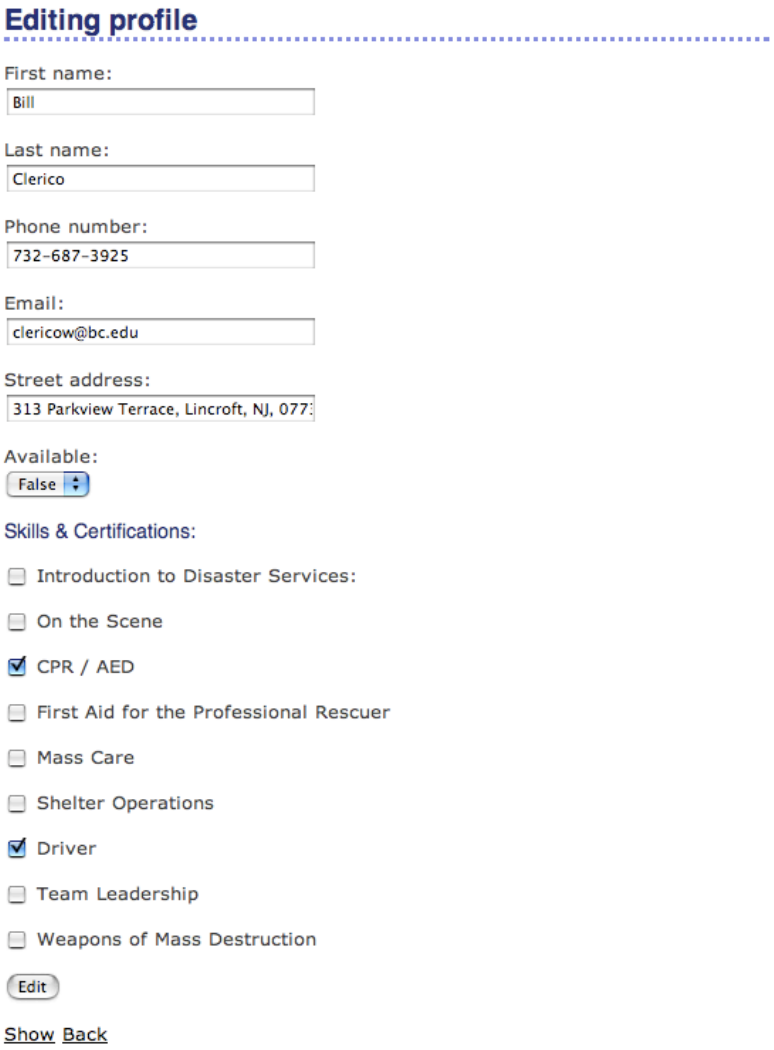
The edit_profile view and corresponding controller method are one of the pivotal pieces of this application. It is through this view that users can input the data which drives the rest of the processes. The first, and most important, information is personal data and contact information. The integrity and format of this data is enforced by regular expressions in the profile model. For example, the email address is validated by the following statement in profile.rb (See code appendix for more details):

```
validates_format_of :email,
        :with => %r{.+@.+\..+}i,
        :message => "must be of the form name@domain.suffix."
```

The edit_profile view also provides the capability to store skills & certifications. These will be used later in the search function.

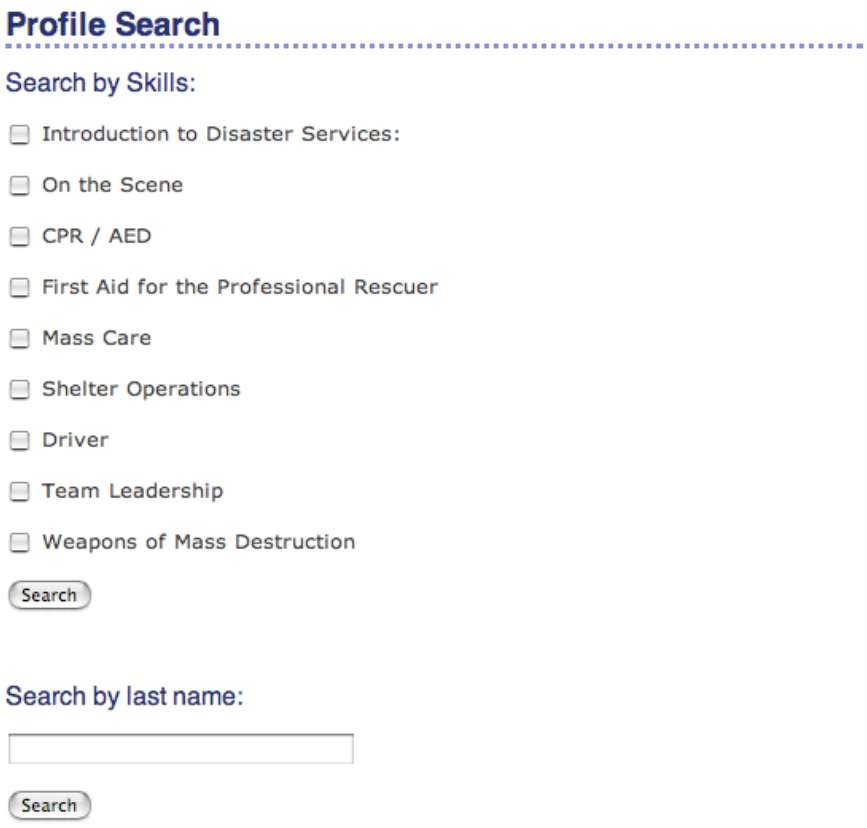
All of this information is written to the database by way of ActiveRecord. The controller merely has to call Object.new and Object.save to create new objects and write them to the database. To retrieve rows, Object.find is called, which can be passed a variety of helpful attributes. Rather than write the SQL queries to create these tables, best RoR practice is to use a database migration. These migrations provide automatic versioning, along

with the capability to roll the database back to previous versions. See the migration code appendix for more details.



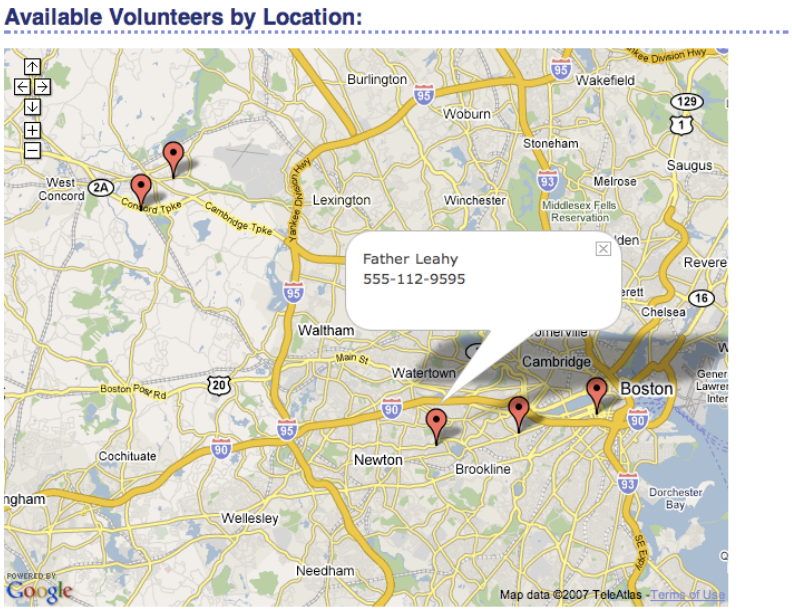
Edit Profile view.

The search view of the portal controller provides access to a series of calls to "find" by the search method. This view allows searches by skill or by last name. After searching, this method calls another view, search2first or search2last to display the results of the query.



Profile search view.

One of the biggest issues facing the red cross was the issue of speeding up response time, and one of the suggestions made was a visual representation of search data.



show_google_map view

While there is much more functionality that this view could benefit from (plotting incidents, filtering results, etc.), this exists as a proof of concept - that a geography based display of volunteers can greatly assist with planning. The process to do this is somewhat lengthy - one must apply for a Google Maps API key, and then write javascript to interface with Google Maps. This javascript is dynamically generated by calls to ruby methods. Furthermore, there is no API ability to map addresses - instead, addresses must be geocoded, and the corresponding longitude and latitude sent to google to be placed on the map. This geocoding can take a few seconds, so rather than geocoding each address every time the map is requested, I geocode the addresses in the :edit_profile method and store these in the database. This had a dramatic increase on performance.

Admin

In addition to the portal controller, there also exists an admin controller, which is designed to be used as a control panel for a site administrator to add and remove users. This control panel is protected by the :authorize_as_admin method, which checks to make sure a user is considered an administrator before permitting access.

CONCLUSION

While this application is by no means a complete implementation of the needs and desires expressed to me by those I interviewed in my case study, it is an attempt to illustrate that these needs can be met through an elegant application of web programming. However, solving these needs is not as simple as merely writing code; creative solutions must be developed to finance, host, and sustain the development of this application. While the architectural underpinnings of Ruby on Rails make this simpler than if other technologies were used, these difficulties still exist and must be addressed.

By keeping with the design principles of Web 2.0: the use of the Internet as a platform, data-centric programming, and harnessing collective intelligence - application design can be more closely married with the problems that it is trying to solve. This marriage will ultimately lead to an increase in efficiency and effectiveness, and a decrease in cost.

BIBLIOGRAPHY

"A Second Century of Service: 2006 Annual Report". *The American Red Cross of Massachusetts Bay.* http://www.bostonredcross.org/your_red_cross/2006AnnualReport.final.pdf

"About Us". *American Red Cross.* http://www.redcross.org/aboutus

Bill Bradley, Paul Jansen, and Les Silverman. "The NonProfit Sector's $100 Billion Opportunity". *The Harvard Business Review,* May 2003.

Drummond, Hugh. Director of External Relations, American Red Cross of Massachusetts Bay. Interview. 11/4/06.

Eric Knoor, Leon Erlanger, and James R. Borck. "A field guide to software as a service". *InfoWorld.* 4/18/05.

Fowler, Chad. *Rails Recipes.* Raleigh: The Pragmatic Bookshelf, 2006.

Gatzke, Ludwig. Web 2.0 Graphic. http://flickr.com/photos/stabilo-boss/

"Introduction to Active Server Pages". *AHref.com.* http://www.ahref.com/guides/technology/199806/0601buz.html

Jacque Bughin and James Mayika. "How businesses are using Web 2.0: A McKinsey Global Survey". *The McKinsey Quarterly,* 2007.

Klau, Michael. Team Leader, American Red Cross of Massachusetts Bay. Interview. 10/17/06.

"Locomotive: Ruby on Rails development done the Mac way". http://locomotive.raaum.org

"Monday Night Team". *American Red Cross.* http://www.arcmb.org

Orsini, Rob. *Rails Cookbook.* Cambridge: O'Reilly, 2007.

Schneir, Bruce. "Cryptanalysis of SHA-1". http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html

Solman, John. North Zone Manager, American Red Cross of Massachusetts Bay. Interview. 3/20/07.

"CRM Software Leader". *Salesforce.com.* http://www.salesforce.com

"The Official Microsoft ASP.NET 2.0 Site". *Microsoft Corporation.* http://www.asp.net

Thomas, Dave. *Agile Web Development with Rails.* Raleigh: The Pragmatic Bookshelf, 2006.

"Using Ruby on Rails for Web Development on Mac OS X". *Apple Developer Connection.* http://developer.apple.com/tools/rubyonrails.html : 6/8/06.

"What You Should Know About Nonprofits". National Center for Nonprofit Boards & Independent Sector. http://www.independentsector.org.

CODE APPENDIX

```
# MySQL (default setup).  Versions 4.1 and 5.0 are recommended.
#
# Install the MySQL driver:
#   gem install mysql
# On MacOS X:
#   gem install mysql -- --include=/usr/local/lib
# On Windows:
#   There is no gem for Windows.  Install mysql.so from RubyForApache.
#   http://rubyforge.org/projects/rubyforapache
#
# And be sure to use new-style password hashing:
#   http://dev.mysql.com/doc/refman/5.0/en/old-client.html
development:
  adapter: mysql
  database: resources_development
  username: root
  password:
  host: localhost

# Warning: The database defined as 'test' will be erased and
# re-generated from your development database when you run 'rake'.
# Do not set this db to the same as development or production.
test:
  adapter: mysql
  database: resources_test
  username: root
  password:
  host: localhost

production:
  adapter: mysql
  database: resources_production
  username: root
  password:
  host: localhost
```

```ruby
class AdminController < ApplicationController
  before_filter :authorize

  def index
    list
    render :action => 'list'
  end

  # GETs should be safe (see http://www.w3.org/2001/tag/doc/whenToUseGet.html)
  verify :method => :post, :only => [ :destroy, :create, :update ],
         :redirect_to => { :action => :list }

  def list
    @profile_pages, @profiles = paginate :profiles, :per_page => 10
  end

  def show
    @profile = Profile.find(params[:id])
  end

  def new
    @profile = Profile.new
  end

  def create
   # @cart = Cart.new
    @profile = Profile.new(params[:profile])
    # @profile.cart = @cart
    if @profile.save
      flash[:notice] = 'Profile was successfully created.'
      redirect_to :action => 'list'
    else
      render :action => 'new'
    end
  end

  def edit
    @profile = Profile.find(params[:id])
  end

  def update
    @profile = Profile.find(params[:id])
    if @profile.update_attributes(params[:profile])
      flash[:notice] = 'Profile was successfully updated.'
      redirect_to :action => 'show', :id => @profile
    else
      render :action => 'edit'
    end
  end

  def destroy
    Profile.find(params[:id]).destroy
    redirect_to :action => 'list'
  end
end
```

```
# Filters added to this controller will be run for all controllers in the application.
# Likewise, all the methods added will be available for all controllers.
class ApplicationController < ActionController::Base

private

def authorize
    unless User.find_by_id(session[:user_id])
        session[:original_uri] = request.request_uri
        flash[:notice] = "Please log in"
        redirect_to(:controller => "login", :action => "login")
    end
end

def authorizeadmin
    @user = User.find_by_id(session[:user_id])

    unless  @user != nil and user.admin = 1
                flash[:notice] = "You must be logged in as an administrator to access this
area."
                redirect_to(:controller => "login", :action => "login")
    end
end

end
```

```ruby
class LoginController < ApplicationController

    before_filter :authorize, :except => [:login, :add_user]
    #before_filter :authorizeadmin, :except => [:login, :logout, :index]

  def add_user

    @user = User.new(params[:user])
    if request.post? and @user.save
        flash.now[:notice] = "User #{@user.name} created"
        redirect_to(:action => "login")
        @user = User.new
    end
  end

  def login
    session[:user_id] = nil
    if request.post?
        user = User.authenticate(params[:name], params[:password])
        if user
            session[:user_id] = user.id
            uri = session[:original_uri]
            session[:original_uri] = nil
            redirect_to(uri || { :controller => 'portal', :action => "index"})
        else
            flash[:notice] = "Invalid user/password combination"
        end
    end


  end

  def logout
    session[:user_id] = nil
    flash[:notice] = "Logged out"
    redirect_to(:action => "login")
  end

  def index
  end

  def delete_user

        user = User.find(params[:id])
        user.destroy
      redirect_to(:action => :list_users)
  end

  def list_users
    @all_users = User.find(:all)
  end
end
```

```
class PortalController < ApplicationController

  #require 'rubygems' # Unless you install from the tarball or zip.
  #require 'icalendar'
  #require 'date'
  #include Icalendar # Probably do this in your class to limit namespace overlap

  before_filter :authorize

  def index
    @currentuser = User.find_by_id(session[:user_id])
    @myprofile = Profile.find_by_id(@currentuser.profileid)
    #flash[:notice] = "Current user name: " + @currentuser.name.to_s + " Profile: " +
@currentuser.profileid.to_s
  end

  def find_user_id
    unless session[:user_id]
        session[:user_id] = Login.login
    end
    session[:user_id]
  end


  def availability
    @profiles = Profile.find_available
    @availprofiles = Profile.find_available
    @currentuser = User.find_by_id(session[:user_id])
  end

  def create
    @profile = Profile.new(params[:profile])
    if @profile.save
      @user = User.find_by_id(session[:user_id])
      if @user.update_attribute(:profileid, @profile.id)
           lookup_geocode(@profile.street_address)
         redirect_to :action => 'index'
      else
        flash[:notice] = "Error updating user.  Please contact technical support."
        render :action => 'new'
      end
    else
      flash[:notice] = "Error saving profile.  Please contact technical support."
      render :action => 'new'
    end
  end

  def edit_profile
    @user = User.find_by_id(session[:user_id])
    #flash[:notice] = "User id: " + @user.id.to_s + " Profile id: " + @user.profileid.to_s
    @profile = Profile.find_by_id(@user.profileid)
    @user.save

  end

   def update
    @profile = Profile.find(params[:id])
    if @profile.update_attributes(params[:profile])
      lookup_geocode(@profile.street_address)
      redirect_to :action => 'show', :id => @profile
```

```
    else
      flash[:notice] = "There was an error updating your profile."
      render :action => 'edit_profile'
    end
  end

  def show
    @profile = Profile.find(params[:id])
  end

  def display
    @profile = Profile.find(params[:od])
  end

  def new
    @profile = Profile.new
  end

  def search
  end

  def search2first
    @profiles = Profile.find(:all, :conditions=> params[:profile])
  end

  def search2last
    @profiles = Profile.find(:all, :conditions=> params[:profile])
  end

  def lookup_geocode(address)
        @currentuser = User.find_by_id(session[:user_id])
        @myprofile = Profile.find_by_id(@currentuser.profileid)
        geocode = get_geocode address

        # geo_code is now a hash with keys :latitude and :longitude
        # place these values back into our "database" (array of hashes)
        @myprofile.latitude = geocode[:latitude]
        @myprofile.longitude = geocode[:longitude]
        @myprofile.save

  end


  def show_google_map
    # all we're going to do is loop through the @places array on the page
        @currentuser = User.find_by_id(session[:user_id])
        @myprofile = Profile.find_by_id(@currentuser.profileid)
        @profiles = Profile.find_available
  end

  private
  def get_geocode(address)
    logger.debug 'starting geocoder call for address: '+address
    # this is where we call the geocoding web service
    server = XMLRPC::Client.new2('http://rpc.geocoder.us/service/xmlrpc')
    result = server.call2('geocode', address)
    logger.debug "Geocode call: "+result.inspect
    if result
        return {:success=> true, :latitude=> result[1][0]['lat'],
              :longitude=> result[1][0]['long']}
```

```
    else
        flash[:notice] = "There was an error looking up your address for GeoCoding.  Please
check it."
    end
  end

end
```

```ruby
class Profile < ActiveRecord::Base

    #attr_reader :id

    def self.find_available
        find_all_by_available(true)
    end

    validates_presence_of :first_name, :last_name, :phone_number, :email, :street_address
    validates_format_of :email,
        :with => %r{.+@.+\..+}i,
        :message => "must be of the form name@domain.suffix."
    validates_format_of :phone_number,
        :with => %r{...-...-....}i,
        :message => "must be of the form xxx-xxx-xxxx."
    validates_format_of :street_address,
        :with => %r{.+,.+,.+}i,
        :message => "must be a proper address of the form: Street, City, State"

end
```

```ruby
class User < ActiveRecord::Base

    validates_presence_of   :name
    validates_uniqueness_of :name

    attr_accessor   :password_confirmation

    validates_confirmation_of   :password

    def validate
        errors.add_to_base("Missing password") if hashed_password.blank?
    end

    def password
        @password
    end

    def password=(pwd)
        @password = pwd
        create_new_salt
        self.hashed_password = User.encrypted_password(self.password, self.salt)
    end

    def prid
        self.profileid
    end

    def prid=(pi)
        self.profileid = pi
    end

    def aid
        self.admin
    end

    def aid=(adid)
        self.admin = adid
    end

    def self.authenticate(name, password)
        user = self.find_by_name(name)
        if user
            expected_password = encrypted_password(password, user.salt)
            if user.hashed_password != expected_password
                user = nil
            end
        end
        user
    end
    #clever active record trick - creates a finder even though we don't define a method

    private

    def self.encrypted_password(password, salt)
        string_to_hash = password + "wibble" + salt
        Digest::SHA1.hexdigest(string_to_hash)
    end

    def create_new_salt
        self.salt = self.object_id.to_s + rand.to_s
```

```
    end

end
```

```
<%= error_messages_for 'profile' %>

<!--[form:profile]-->
<p><label for="profile_first_name">First name</label><br/>
<%= text_field 'profile', 'first_name'  %></p>

<p><label for="profile_last_name">Last name</label><br/>
<%= text_field 'profile', 'last_name'  %></p>

<p><label for="profile_phone_number">Phone number</label><br/>
<%= text_field 'profile', 'phone_number'  %></p>

<p><label for="profile_email">Email</label><br/>
<%= text_field 'profile', 'email'  %></p>

<p><label for="profile_street_address">Street address</label><br/>
<%= text_field 'profile', 'street_address'  %></p>

<p><label for="profile_available">Available</label><br/>
<select id="profile_available" name="profile[available]"><option value="false">False</
option><option value="true">True</option></select></p>
<!--[eoform:profile]-->
```

```
<h1>Editing profile</h1>

<%= start_form_tag :action => 'update', :id => @profile %>
  <%= render :partial => 'form' %>
  <%= submit_tag 'Edit' %>
<%= end_form_tag %>

<%= link_to 'Show', :action => 'show', :id => @profile %>
<%= link_to 'Back', :action => 'list' %>
```

```
<h1>Listing profiles</h1>

<table>
  <tr>
  <% for column in Profile.content_columns %>
    <th><%= column.human_name %></th>
  <% end %>
  </tr>

<% for profile in @profiles %>
  <tr>
  <% for column in Profile.content_columns %>
    <td><%=h profile.send(column.name) %></td>
  <% end %>
    <td><%= link_to 'Show', :action => 'show', :id => profile %></td>
    <td><%= link_to 'Edit', :action => 'edit', :id => profile %></td>
    <td><%= link_to 'Destroy', { :action => 'destroy', :id => profile }, :confirm => 'Are
you sure?', :post => true %></td>
  </tr>
<% end %>
</table>

<%= link_to 'Previous page', { :page => @profile_pages.current.previous } if
@profile_pages.current.previous %>
<%= link_to 'Next page', { :page => @profile_pages.current.next } if
@profile_pages.current.next %>

<br />

<%= link_to 'New profile', :action => 'new' %>
```

```
<h1>New profile</h1>

<%= start_form_tag :action => 'create' %>
  <%= render :partial => 'form' %>
  <%= submit_tag "Create" %>
<%= end_form_tag %>

<%= link_to 'Back', :action => 'list' %>
```

```
<% for column in Profile.content_columns %>
<p>
  <b><%= column.human_name %>:</b> <%=h @profile.send(column.name) %>
</p>
<% end %>

<%= link_to 'Edit', :action => 'edit', :id => @profile %> |
<%= link_to 'Back', :action => 'list' %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!--
 ! Excerpted from "Agile Web Development with Rails, 2nd Ed."
 ! We make no guarantees that this code is fit for any purpose.
 ! Visit http://www.pragmaticprogrammer.com/titles/rails2 for more book information.
-->
<html>
<head>
  <title>Administration</title>
  <%= stylesheet_link_tag "scaffold", "depot", :media => "all" %>
</head>
<body id="admin">
  <div id="banner">
    <!--<img src="/images/logo.png"/>-->
    <%= @page_title || "Administrator's Panel" %>
  </div>
  <div id="columns">
    <div id="side">
      <p>
       <!--<%= link_to "Products",   :controller => 'admin', :action => 'list' %>-->
      </p>
      <p>
        <%= link_to "List users", :controller => 'login', :action => 'list_users' %>
        <br/>
        <%= link_to "Add user",   :controller => 'login', :action => 'add_user' %>
      </p>
      <p>
        <%= link_to "Logout",     :controller => 'login', :action => 'logout' %>
      </p>
    </div>
    <div id="main">
    <% if flash[:notice] -%>
      <div id="notice"><%= flash[:notice] %></div>
    <% end -%>
      <%= yield :layout %>
    </div>
  </div>
</body>
</html>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                      "http://www.w3.org/TR/xhtml1/DTD/xhmtl1-transitional.dtd">

<html>
<head>
  <title>Login: <%= controller.action_name %></title>
  <%= stylesheet_link_tag 'scaffold', 'depot' %>
</head>
<body>

<p style="color: green"><%= flash[:notice] %></p>

<%= yield  %>

</body>
</html>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!--
 ! Excerpted from "Agile Web Development with Rails, 2nd Ed."
 ! We make no guarantees that this code is fit for any purpose.
 ! Visit http://www.pragmaticprogrammer.com/titles/rails2 for more book information.
-->
<html>
<head>
  <title>Portal</title>
  <%= stylesheet_link_tag "scaffold", "depot", :media => "all" %>
</head>
<body id="admin">
  <div id="banner">
    <!--<img src="/images/logo.png"/>-->
    <%= @page_title || "Welcome to the Volunteer Portal" %>
  </div>
  <div id="columns">
    <div id="side">
      <p>
        <%= link_to "Home", :controller => 'portal', :action => 'index' %>
      </p>
      <p>
        <%= link_to "Availability", :controller => 'portal', :action => 'availability' %>
      </p>
      <p>
         <%= link_to "Edit Profile", :controller => 'portal', :action => 'edit_profile' %>
      </p>
      <p>
        <%= link_to "Search",     :controller => 'portal', :action => 'search' %>
      </p>
      <p>
        <%= link_to "Map",     :controller => 'portal', :action => 'show_google_map' %>
      </p>
      <p>
        <%= link_to "Logout",     :controller => 'login', :action => 'logout' %>
      </p>
    </div>
    <div id="main">
    <% if flash[:notice] -%>
      <div id="notice"><%= flash[:notice] %></div>
    <% end -%>
      <%= yield :layout %>
    </div>
  </div>
</body>
</html>
```

```
<div class="resources-form">
    <%= error_messages_for 'user' %>

    <fieldset>
        <legend>Enter User Details</legend>

        <% form_for :user do |form| %>
            <p>
                <label for="user_name">User Name:</label>
                <%= form.text_field :name, :size =>40 %>
            </p>

            <p>
                <label for="user_password">Password:</label>
                <%= form.password_field :password, :size =>40 %>
            </p>

            <p>
                <label for="user_name">Confirm Password:</label>
                <%= form.password_field :password_confirmation, :size =>40 %>
            </p>

            <%= submit_tag "Add User", :class => "submit" %>

        <% end %>
    </fieldset>
</div>
```

```
<h1>Welcome to the American Red Cross Volunteer Portal</h1>

<% if @myprofile -%>
    Hello, <%= h(@myprofile.first_name) %>.
    <%if @myprofile.available = true -%>
        You are currently available.
    <% else -%>
        You are currently unavailable.
    <% end %>
<% end %>



<BR><BR><HR>

<h2>Availability:</h2>
<% for profile in @availprofiles -%>
    <div class="entry">
        <h3><%= h(profile.last_name)%>, <%= h(profile.first_name) %></h3>
        <%= profile.phone_number %>
    </div>
<% end %>

<HR>

<%= link_to "Edit my profile", :controller => 'portal', :action => 'edit_profile' %>
```

```
<h1>Users</h1>
<ul>
  <% for user in @all_users %>
    <li><%= link_to "[X]", { # link_to options
                             :controller => 'login',
                             :action => 'delete_user',
                             :id => user},
                           { # html options
                             :method  => :post,
                             :confirm => "Really delete #{user.name}?"
                           } %>
        <%= h(user.name) %>
    </li>
  <% end %>
</ul>
```

```
<h1>Welcome to the Volunteer Portal</h1>

<div class="depot-form">
  <fieldset>
    <legend>Please Log In</legend>

    <%= form_tag %>
      <p>
        <label for="name">Name:</label>
        <%= text_field_tag :name, params[:name] %>
      </p>

      <p>
        <label for="password">Password:</label>
        <%= password_field_tag :password, params[:password] %>
      </p>

      <p>
        <%= submit_tag "Login" %>
      </p>
    <%= end_form_tag %>
  </fieldset>
</div>
<BR>
Don't have an account?  <%= link_to 'Click here', :action => 'add_user' %> to create one.
```

```
<h1>Login#logout</h1>
<p>Find me in app/views/login/logout.rhtml</p>
```

```
<%= error_messages_for 'profile' %>

<!--[form:profile]-->
<p><label for="profile_first_name">First name</label><br/>
<%= text_field 'profile', 'first_name'  %></p>

<!--[eoform:profile]-->
```

```
<%= error_messages_for 'profile' %>


<!--[form:profile]-->
<p><label for="profile_first_name">First name:</label><br/>
<%= text_field 'profile', 'first_name'  %></p>

<p><label for="profile_last_name">Last name:</label><br/>
<%= text_field 'profile', 'last_name'  %></p>

<p><label for="profile_phone_number">Phone number:</label><br/>
<%= text_field 'profile', 'phone_number'  %></p>

<p><label for="profile_email">Email:</label><br/>
<%= text_field 'profile', 'email'  %></p>

<p><label for="profile_street_address">Street address:</label><br/>
<%= text_field 'profile', 'street_address'  %></p>

<p><label for="profile_available">Available:</label><br/>
<select id="profile_available" name="profile[available]"><option value="false">False</
option><option value="true">True</option></select></p>

<h3>Skills & Certifications:</h3>

    <p>
        <%= check_box 'profile', 'introtodisasterservices'  %>
        <label for="profile_introtodisasterservices">Introduction to Disaster Services:</
label>
    </p>

    <p>
        <%= check_box 'profile', 'onthescene'  %>
        <label for="profile_onthescene">On the Scene</label>
    </p>

    <p>
        <%= check_box 'profile', 'cpraed'  %>
        <label for="profile_cpraed">CPR / AED</label>
    </p>

    <p>
        <%= check_box 'profile', 'firstaid'  %>
        <label for="profile_firstaid">First Aid for the Professional Rescuer</label>
    </p>

    <p>
        <%= check_box 'profile', 'masscare'  %>
        <label for="profile_masscare">Mass Care</label>
    </p>

    <p>
        <%= check_box 'profile', 'shelterops'  %>
        <label for="profile_shelterops">Shelter Operations</label>
    </p>

    <p>
        <%= check_box 'profile', 'driver'  %>
        <label for="profile_driver">Driver</label>
    </p>
```

```
    <p>
        <%= check_box 'profile', 'teamleadership'  %>
        <label for="profile_teamleadership">Team Leadership</label>
    </p>

    <p>
        <%= check_box 'profile', 'wmd'  %>
        <label for="profile_wmd">Weapons of Mass Destruction</label>
    </p>

<!--[eoform:profile]-->
```

```
<h1>Available Profiles:</h1>

<% for profile in @availprofiles -%>
    <div class="entry">
        <h3><%= h(profile.last_name)%>, <%= h(profile.first_name) %></h3>
        <%= profile.phone_number %>
    </div>
<% end %>



<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<BR>



<HR>

<%= link_to "Edit my profile", :controller => 'portal', :action => 'edit_profile' %>
```

```
<H1> Profile creation</h1>

<% if @profile -%>
    Your profile says your first name is <%= h(@profile.first_name) %>.<BR>
    Your profile id is: <%= h(@profile.id.to_s) %>.
<% else -%>
    Your profile is not valid.
<% end %>
```

```
<% for column in Profile.content_columns %>
<p>
  <b><%= column.human_name %>:</b> <%=h @profile.send(column.name) %>
</p>
<% end %>
```

```
<h1>Editing profile</h1>

<%= start_form_tag :action => 'update', :id => @profile %>
  <%= render :partial => 'form' %>
  <%= submit_tag 'Edit' %>
<%= end_form_tag %>

<%= link_to 'Show', :action => 'show', :id => @profile %>
<%= link_to 'Back', :action => 'list' %>
```

```
<h1>Welcome to the American Red Cross Volunteer Portal</h1>

<% if @myprofile -%>
    Hello, <%= h(@myprofile.first_name) %>.
    <!--Available: <%= h(@myprofile.available) %>-->
    <%if @myprofile.available == true -%>
        You are currently available.
    <% else -%>
        You are currently unavailable.
    <% end %>
<% else -%>
    You haven't created a profile yet! <BR>
    <%= link_to "Click here", :controller => 'portal', :action => 'new' %> to create a
profile.
<% end %>



<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<BR>



<HR>
<%= link_to "Edit my profile", :controller => 'portal', :action => 'edit_profile' %>
```

```
<h1>New profile</h1>

<%= start_form_tag :action => 'create' %>
  <%= render :partial => 'form' %>
  <%= submit_tag "Create" %>
<%= end_form_tag %>

<%= link_to 'Back', :action => 'list' %>
```

```
<h1>Profile Search</h1>

<%= start_form_tag :action => 'search2first' %>

<h2>Search by Skills:</h2>
    <p>
        <%= check_box 'profile', 'introtodisasterservices'  %>
        <label for="profile_introtodisasterservices">Introduction to Disaster Services:</
label>
    </p>

    <p>
        <%= check_box 'profile', 'onthescene'  %>
        <label for="profile_onthescene">On the Scene</label>
    </p>

    <p>
        <%= check_box 'profile', 'cpraed'  %>
        <label for="profile_cpraed">CPR / AED</label>
    </p>

    <p>
        <%= check_box 'profile', 'firstaid'  %>
        <label for="profile_firstaid">First Aid for the Professional Rescuer</label>
    </p>

    <p>
        <%= check_box 'profile', 'masscare'  %>
        <label for="profile_masscare">Mass Care</label>
    </p>

    <p>
        <%= check_box 'profile', 'shelterops'  %>
        <label for="profile_shelterops">Shelter Operations</label>
    </p>

    <p>
        <%= check_box 'profile', 'driver'  %>
        <label for="profile_driver">Driver</label>
    </p>

    <p>
        <%= check_box 'profile', 'teamleadership'  %>
        <label for="profile_teamleadership">Team Leadership</label>
    </p>

    <p>
        <%= check_box 'profile', 'wmd'  %>
        <label for="profile_wmd">Weapons of Mass Destruction</label>
    </p>

  <%= submit_tag "Search" %>
<%= end_form_tag %>



<BR>


<%= start_form_tag :action => 'search2last' %>
```

```
    <p>
        <h2><label for="profile_last_name">Search by last name:</label></h2>
        <%= text_field 'profile', 'last_name'  %>
    </p>

  <%= submit_tag "Search" %>
<%= end_form_tag %>


<BR><BR>



<HR>
```

```
<h1>Profile Search</h1>

<table>
  <tr>
  <% for column in Profile.content_columns %>
    <th><%= column.human_name %></th>
    <% if column.human_name == "Street address" %>
         <% break %>
    <% end %>
  <% end %>
  </tr>

<% for profile in @profiles %>
  <tr>
  <% for column in Profile.content_columns %>

    <td><%=h profile.send(column.name) %></td>

        <% if column.human_name == "Street address" %>
         <% break %>
    <% end %>
  <% end %>
  </tr>
<% end %>
</table>


<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>



<HR>
```

```
<h1>Profile Search</h1>

<table>
  <tr>
  <% for column in Profile.content_columns %>
    <th><%= column.human_name %></th>
  <% end %>
  </tr>

<% for profile in @profiles %>
  <tr>
  <% for column in Profile.content_columns %>
    <td><%=h profile.send(column.name) %></td>
  <% end %>
  </tr>
<% end %>
</table>


<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<BR>



<HR>
```

```
<html>
<head>
  <title>Map</title>

    <h1> Available Volunteers by Location:</h1>

    <!-- This includes the google maps API code.
      You need to put your own key here -->
    <script src="http://maps.google.com/maps?
file=api&v=2&key=ABQIAAAA5UUmMkZKOQxTN7jPWRhQyRRd_f1wgoxrA90ATXKnKNuxhfd85BRu4K2Aug1ssstNY_g
nKS7HiOTrHA"
    type="text/javascript"></script>

    <script type="text/javascript">
        // helper function to create markers
        function createMarker(point,html) {
            var marker = new GMarker(point);
            GEvent.addListener(marker, "click", function() {
             marker.openInfoWindowHtml(html);
             });
            return marker;
        }

        // this is called when the page loads.
        // it initializes the map, and creates each marker
        function initialize() {
            var map = new GMap(document.getElementById("map"));
            map.addControl(new GSmallMapControl());

            var point = new GPoint(<%=@myprofile.longitude%>,<%=@myprofile.latitude%>);
            map.centerAndZoom(point, 7);
            var marker = createMarker(point,'<div>Me!</div>')
            map.addOverlay(marker);

            <% for profile in @profiles %>
                <% if profile.id != @myprofile.id %>
                    var point = new GPoint(<%=profile.longitude%>,<%=profile.latitude%>);
                    var marker = createMarker(point,'<div><%=h profile.first_name%> <%= h
profile.last_name%> <BR> <%= h profile.phone_number%></div>');
                    map.addOverlay(marker);
                <% end %>
            <% end %>

        }
    </script>

</head>
<body onload="initialize()">

<!-- This is the element in which the map will be displayed. -->
<div id="map" style="width: 650px; height: 500px"></div>


</body>
</html>
```

```
<% for column in Profile.content_columns %>
<p>
  <b><%= column.human_name %>:</b> <%=h @profile.send(column.name) %>
</p>
<% end %>

<%= link_to 'Edit', :action => 'edit', :id => @profile %> |
<%= link_to 'Back', :action => 'list' %>
```

```ruby
class CreateProfiles < ActiveRecord::Migration
  def self.up
    create_table :profiles do |t|
      t.column :first_name,     :string
      t.column :last_name,      :string
      t.column :phone_number,   :string
      t.column :email,          :string
    end
  end

  def self.down
    drop_table :profiles
  end
end
```

```ruby
class AddAddress < ActiveRecord::Migration
  def self.up
    add_column :profiles, :street_address, :string
  end

  def self.down
    remove_column :profiles, :street_address
  end
end
```

```ruby
class AddTestData < ActiveRecord::Migration
  def self.up
    Profile.create(:first_name => 'John',
        :last_name => 'Doe',
        :phone_number => '617-555-1212',
        :email => 'jdoe@microsoft.com',
        :street_address => '123 Pleasant Street',
        :available => true)
    Profile.create(:first_name => 'James',
        :last_name => 'Smith',
        :phone_number => '732-342-392',
        :email => 'jsmith@apple.com',
        :street_address => '123 Commonwealth Avenue',
        :available => true)
    Profile.create(:first_name => 'Marisa',
        :last_name => 'Tester',
        :phone_number => '507-321-6677',
        :email => 'mtester@hp.com',
        :street_address => '99 Infinite Loop',
        :available => false)
    Profile.create(:first_name => 'Father',
        :last_name => 'Leahy',
        :phone_number => '617-655-6392',
        :email => 'leahy@bc.edu',
        :street_address => '26 Campanella Way',
        :available => false)
  end

  def self.down
    Profile.delete_all
  end
end
```

```ruby
class AddAvailability < ActiveRecord::Migration
  def self.up
    add_column :profiles, :available, :boolean
  end

  def self.down
    remove_column :profiles, :available
  end
end
```

```ruby
class CreateUsers < ActiveRecord::Migration
  def self.up
    create_table :users do |t|
      t.column :name, :string
      t.column :hashed_password, :string
      t.column :salt, :string
    end
  end

  def self.down
    drop_table :users
  end
end
```

```ruby
class AddSessions < ActiveRecord::Migration
  def self.up
    create_table :sessions do |t|
      t.column :session_id, :string
      t.column :data, :text
      t.column :updated_at, :datetime
    end

    add_index :sessions, :session_id
  end

  def self.down
    drop_table :sessions
  end
end
```

```ruby
class AddProfileToUser < ActiveRecord::Migration
  def self.up
    add_column :users, :profileid, :integer
  end

  def self.down
      remove_column :users, :profileid
  end
end
```

```ruby
class AddAdminToUser < ActiveRecord::Migration
  def self.up
    add_column :users, :admin, :boolean
  end

  def self.down
      remove_column :users, :admin
  end
end
```

```
class CreateCart < ActiveRecord::Migration
  def self.up
    create_table :friends do |t|
        t.column :friendid, :integer
    end
  end

  def self.down
    drop_table :friends
  end
end
```

```ruby
class CreateCarts < ActiveRecord::Migration
  def self.up
    create_table :carts do |t|
      # t.column :name, :string
    end
  end

  def self.down
    drop_table :carts
  end
end
```

```ruby
class AddGeocode < ActiveRecord::Migration
  def self.up
    add_column :profiles, :longitude, :decimal, :precision => 9, :scale=> 6
    add_column :profiles, :latitude, :decimal, :precision => 9, :scale=> 6
  end

  def self.down
    remove_column :profiles, :longitude
    remove_column :profiles, :latitude
  end
end
```

```ruby
class AddSkills < ActiveRecord::Migration
  def self.up
    add_column :profiles, :introtodisasterservices, :boolean, :default=> false
    add_column :profiles, :onthescene, :boolean, :default=> false
    add_column :profiles, :cpraed, :boolean, :default=> false
    add_column :profiles, :firstaid, :boolean, :default=> false
    add_column :profiles, :masscare, :boolean, :default=> false
    add_column :profiles, :shelterops, :boolean, :default=> false
    add_column :profiles, :driver, :boolean, :default=> false
    add_column :profiles, :teamleadership, :boolean, :default=> false
    add_column :profiles, :wmd, :boolean, :default=> false
  end

  def self.down
    remove_column :profiles, :introtodisasterservices
    remove_column :profiles, :onthescene
    remove_column :profiles, :cpraed
    remove_column :profiles, :firstaid
    remove_column :profiles, :masscare
    remove_column :profiles, :shelterops
    remove_column :profiles, :driver
    remove_column :profiles, :teamleadership
    remove_column :profiles, :wmd
  end
end
```