# MUSICIAN WEB-SERVICE USING RUBY-ON-RAILS, SOAP, FLEX & AJAX

**John A. Dion***
**M.S. in Computer Science, Rivier College 2006**

**Keywords***: musician management application*

## Abstract

*This project at the core is a web service enabling musicians to communicate with one another and with their fans. It is written in an MVC style with Ruby-on-Rails. The primary user interface was written in Adobe Flex. These choices are based generally on practicality for the user but turn out to be very powerful. The main requirement is an active Internet connection. The use of larger data-packages is being considered. This will allow off-line editing with connections only made during a commit. The users may also interact with the system via email as necessary.*

## 1 Introduction

For many years I have sought a solution to the complexities involved in managing a group of musicians. I registered the musicshowcase.com domain back in 1998 [1]. In 1999 I had a working site, written in PHP, that was more like a personal message board for each band. By 2000 I added a VoiceXML application that allowed bands to post show dates and have the text read by a computer when a fan called from a phone. Fans could also call up the number, say the name of the band, and hear about upcoming shows. A lot has changed since then, but the concept is still the same.

This project is the fruit of my labors during the spring 2006 semester. A few of the ideas presented I came up with back in 2000. Primarily, these are the email-based extensions that are currently being investigated from a security perspective.

### 1.1 Objective

The goal of this software project is to provide an *easy and flexible* way for managing music careers. The requirements are twofold: Musicians would like to connect to their website, fan base, and management from anywhere in the world, while they are on tour. Management would like robust software that can connect them to all of their Artists, Distributors and Agents from a single program. Additionally, this program should allow easy updates to artist websites and promotional/sales websites.

### 1.2 Marketing Feasibility

The feasibility of marketing this product is taken from discussions of portions of this idea with various bands and individual musicians over the years. It will require a unified approach using a “one-stop-shop” style. This product should handle the majority of music promotion activities. The core data activities of a band are as follows:



ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).

I) Send and Receive communications with fans (email and snail mail).
II) Custom design their website and/or use some provided templates.
III) Reflect updates to the calendar in all connecting components.
IV) Schedule automatic sending of communications.
V) Add music to their website.
VI) Add an interface to their website from their music compact disc.
VII) Print flyers.
VIII) Schedule gigs.
IX) Respond automatically to email.

To compete with all the diverse software that bands currently use, it would be ideal to implement as many of these features as possible.

### 1.3 Benefits for Users

The benefits for users are as follows:

I) The band saves a lot of time by having all their data integrated in one package.
II) The fan is sure to receive timely and up-to-date information from the band.
III) The band and/or management will save money by being able to handle a lot of the details themselves.
IV) The band will be easily connected with other bands in the same genre and the fan can be informed of bands they might be interested in checking out. This is commonplace marketing for small bands.

### 1.4 Environment

Everything takes place on the web. Connections for the band client are made from an Adobe Flash 9 enabled web browser. The core components are: (1) a MySQL Database [2], (2) a Web-service written in Ruby-on-Rails [3], and (3) a Flash-based user interface written in Adobe Flex 2 [4].

### 1.5 Approach

The design is highly dependent on the MVC [see Glossary] architecture. A benefit of coding with Ruby-on-Rails is the enforcement of this approach. Ruby is a purely object-oriented approach. Everything is an object, including numbers. An example of this would be obtaining the time one minute from now, written as 1.minute_from_now. The programming is fairly straightforward because the code is clearly separated based on models, views, and controllers. The Object-relational Mapping (ORM) approach makes for very easy data manipulation following the CRUD style [see Glossary].



ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).

## 2 Functional Specification

Level 1 Functionality – This Project

I. Calendar: Additions and changes to the calendar updates all necessary modules for the website and schedule sending out automated email updates to fans.
II. Mail: Standard messages sent and received between individuals and within groups.
III. Management of multiple websites on different physical servers.

Level 2 Functionality

IV. Multiple Artist Management: Manage multiple artists from a desktop client that connects to a remote server.
V. Contacts: The standard run-of-the-mill address book.
VI. Style & Design Module: A basic design module that accepts any website design.
VII. Agent and Sales Staff: Additional functionality requests made by management.
VIII. Forum: a forum that allows text and audio.
IX. Audio streaming.
X. Printing and sending flyers

Future Versions

- Stateless Email application data transfer.
- Mobile phone / PDA.
- Messages sent from phone are posted on the website (text and voice).

### 2.1 Case Study

In one of my previous bands, I was in charge of handling all communications via the web, phone, email, and standard mail. It was actually quite time-consuming. I had a program to make updates to the website, a program to transfer the data to the website, and a separate email client set up to send out mailings to the fans. I had to manually add people to the email list. Generally, people would sign up at gigs, or just visit our website and send an email to us. I used another program to design and type up flyers. I needed clip art to make the site and flyers less bland. It was quite tedious.

### 2.2 Required Functionality

In addition to what is specified in the functional spec, there are major security requirements. The majority of the time spent in coding this application was creating a secure, session based login manager. I need to add some state to compensate for the stateless HTTP protocol. A session manager was designed that handles the task without putting undue load on the database and server. Once a person is authenticated, then all calls to the service pass through the session manager. The session ID is a unique random key that is stored in the database and deleted upon closure of the session. No cookies are used. The session is validated on every call to the service to confirm that the user has access to that data. Only



ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).

data that a user is authorized to modify may be modified. The complexity of the task is a consequence of allowing many bands to connect through the same database simultaneously.

The complexity is further exacerbated by the need to allow the client run on machines other than the host machine. This Flash-based client will run only from authenticated domains. On the server side is a crossdomain.xml that restricts connections from clients that have not registered.

### 2.3 Extended Functionality

The client is capable of being coded with multiple user-interfaces. A DLL file was compiled from the web-service WSDL file early on. A quick feasibility test was done on the login manager to confirm the viability of the approach.

## 3 System Requirements

### 3.1 Server-side

- MySQL database.
- Ruby-on-Rails capable web hosting provider.
- A lighttpd server with Fast-CGI.
- Server RAM and CPU is handled by any standard shared hosting provider.

### 3.2 Client-side

- Any browser capable of running Adobe Flash™ version 9.

### 3.3 Additional Requirements

- A high-speed Internet connection.

## 4 Implementation

*Server Side*
A Web-service coded in Ruby-on-Rails that promotes MVC, CRUD, ORM and testing [see Glossary for all definitions]. Coding in Rails starts with the core CRUD and adds to the design as it develops. After writing the core CRUD that connect through simple html forms, some basic testing is done to make that data manipulation is working as intended.

*Client Side*
An Adobe Flex-based client that can be incorporated into any website. When someone signs up for the service the *crossdomain.xml* file is dynamically updated to allow for access to the Web service from the user's selected domain.

### Core Modules

- Calendar: Updates to this calendar can update all necessary modules for the website and schedule sending out automated email updates to fans.
- Mail: Standard messages sent and received to individuals and/or a list.



ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).

- Contacts: The standard run-of-the-mill address book.
- Security.

## 4.1 Use Case Scenario

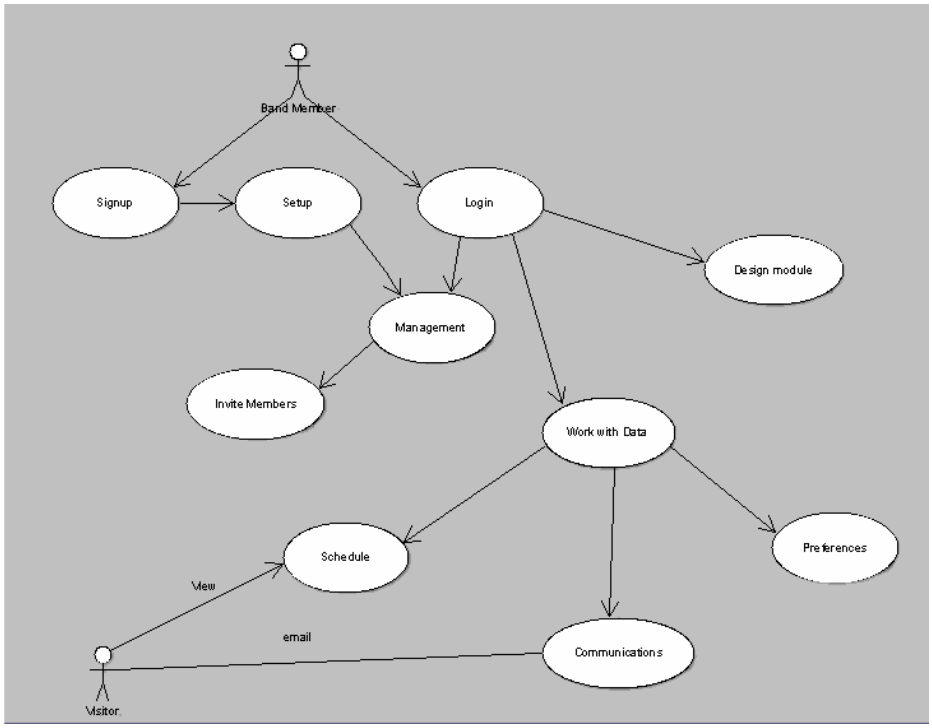


Figure 4.1.1: Use-Case Scenario.

## 4.2 User-interface State Chart

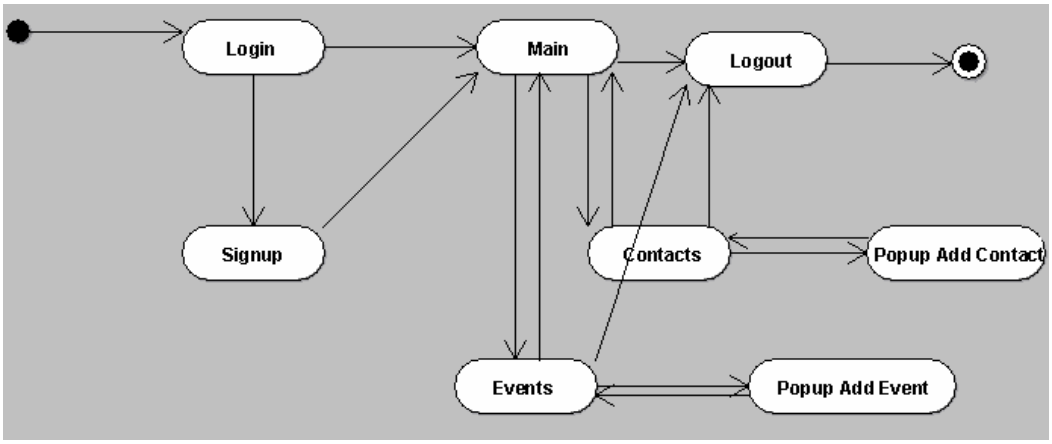


Figure 4.2.1: Use-Interface State Chart.


ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).

## 4.3 Web-service Functions

*Core API Methods*

**member_remove –** *implemented only in non-flex application*
Allows the administrator to remove the member.
**member_add –** *implemented in all*
Add a member is handled during signup.
**member_view –** *implemented in all*
Individual member details, not viewable by user.
**member_edit –** *implemented only in non-flex application for security reasons*
**member_login –** *implemented in all*
Login information, not viewable by user.
**member_logout –** *implemented in all*
Logout call, session closed.
**get_band_id –** *implemented in all*
Band id associated with this user, not viewable by user.
**contact_remove –** *implemented in all*
Contact removed by user id after being authenticated against the member id.
**contact_add –** *implemented in all*
Contact added and associated with band id.
**contact_view –** *implemented in all*
Contact viewed only after being authenticated against the member id.
**contact_list –** *implemented in all*
Contacts viewed only after being authenticated against the member id.
**contact_edit –** *implemented in all*
Contact edited by user id after being authenticated against the member id.
**gig_remove –** *implemented in all*
Event removed after being authenticated against the member id.
**gig_add –** *implemented in all*
Event added and associated with band id.
**gig_view –** *implemented in all*
Event viewed after being authenticated against the member id.
**gig_edit –** *implemented in all*
Event edited after being authenticated against the member id.
**gig_list –** *implemented in all*
Events viewed after being authenticated against the member id.
**job_remove –** *implemented only in non-flex application for security reasons*
**job_add –** *implemented only in non-flex application for security reasons*
**job_view –** *implemented only in non-flex application for security reasons*
**job_edit –** *implemented only in non-flex application for security reasons.*

## 4.4 Implementation Details

A SOAP-based remote procedure call is made from the client base on a user action. All data is passed through the web-service API called bandmanager_api.rb. The implementation for the bandmanager_api



ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).

web-service can be found in bandmanager_controller.rb. These are the two core files for the client side. The controller maps all data to the appropriate data in the database. The request is returned to the client in a SOAP-based XML packet. The client side is heavily-involved in data binding for straight calls to the web service. However, more complex calls such as edits are passed handled by the datahandler functions in the musicshowcase.mxml Flex file.

## 4.5 Screen Shots

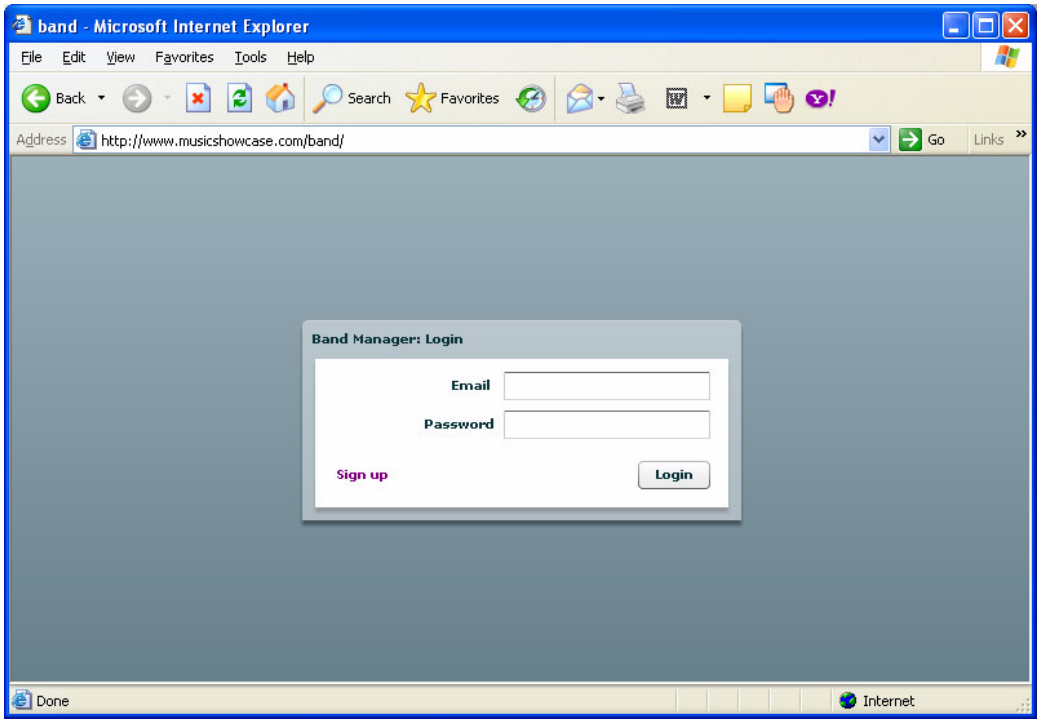


Figure 4.5.1: Login Screen


ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).

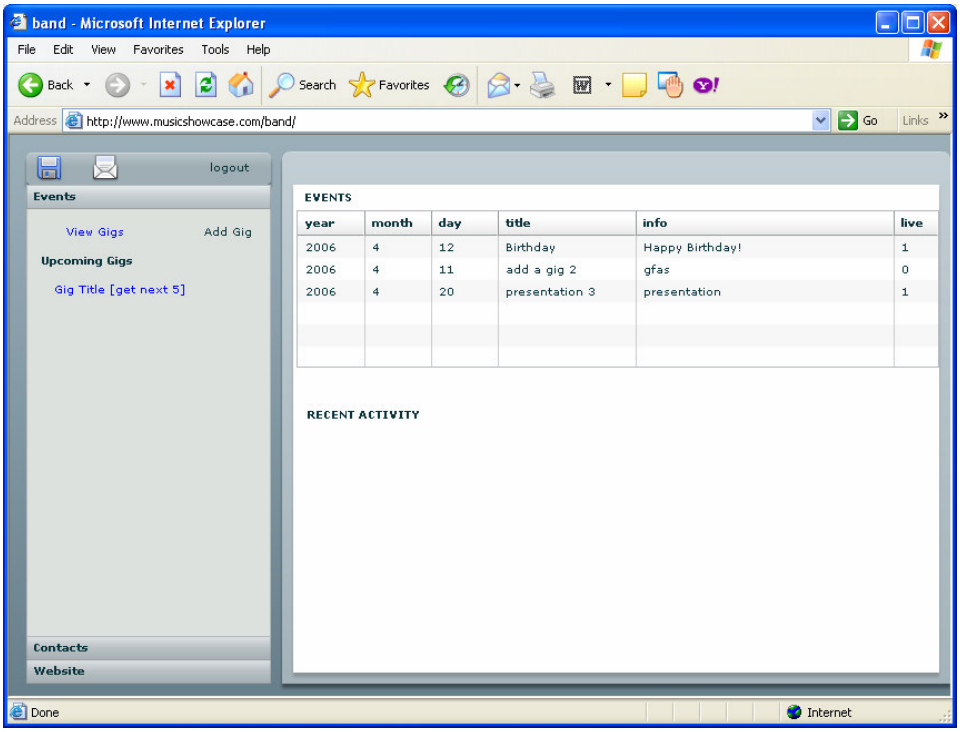


Figure 4.5.2: Main Screen (Events).

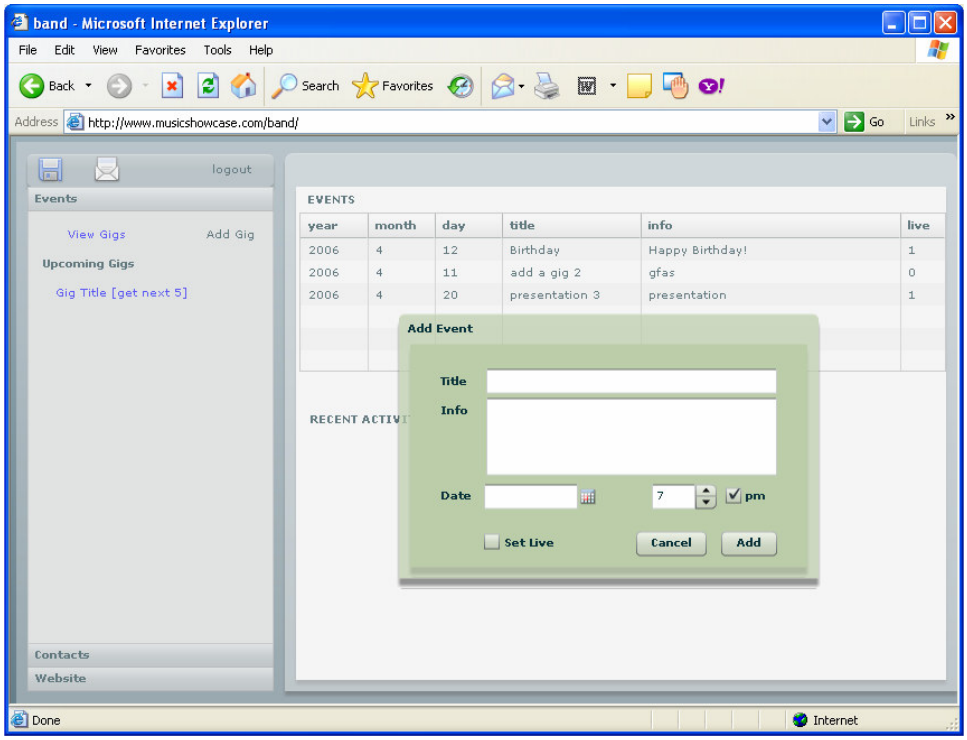


Figure 4.5.3: Add Event Screen.


ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).

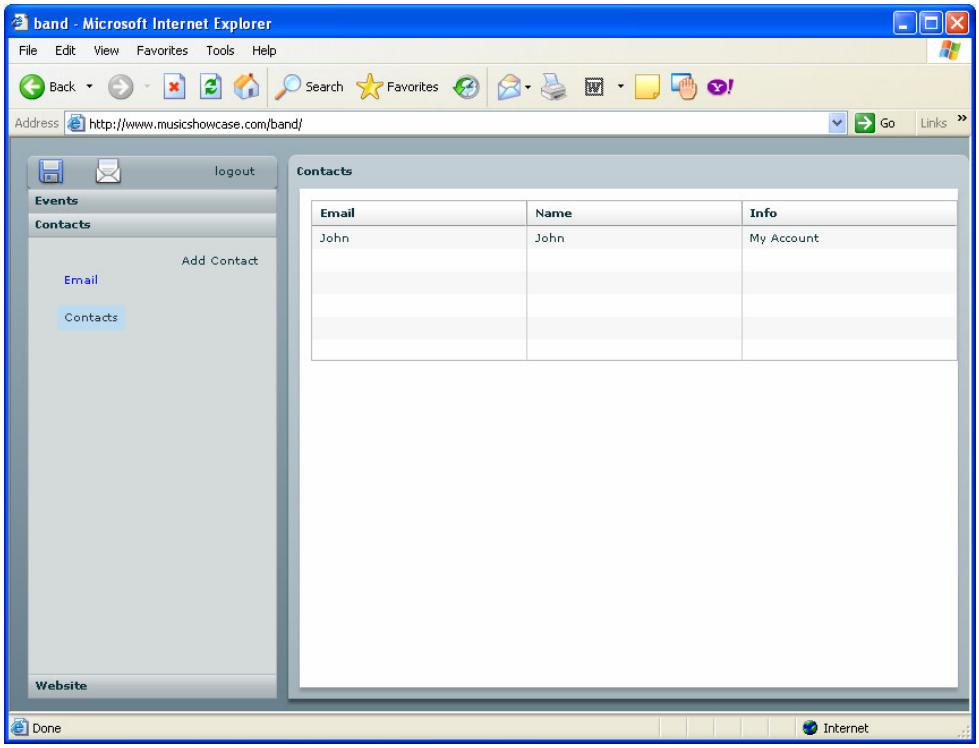


Figure 4.5.4: Contacts Screen.

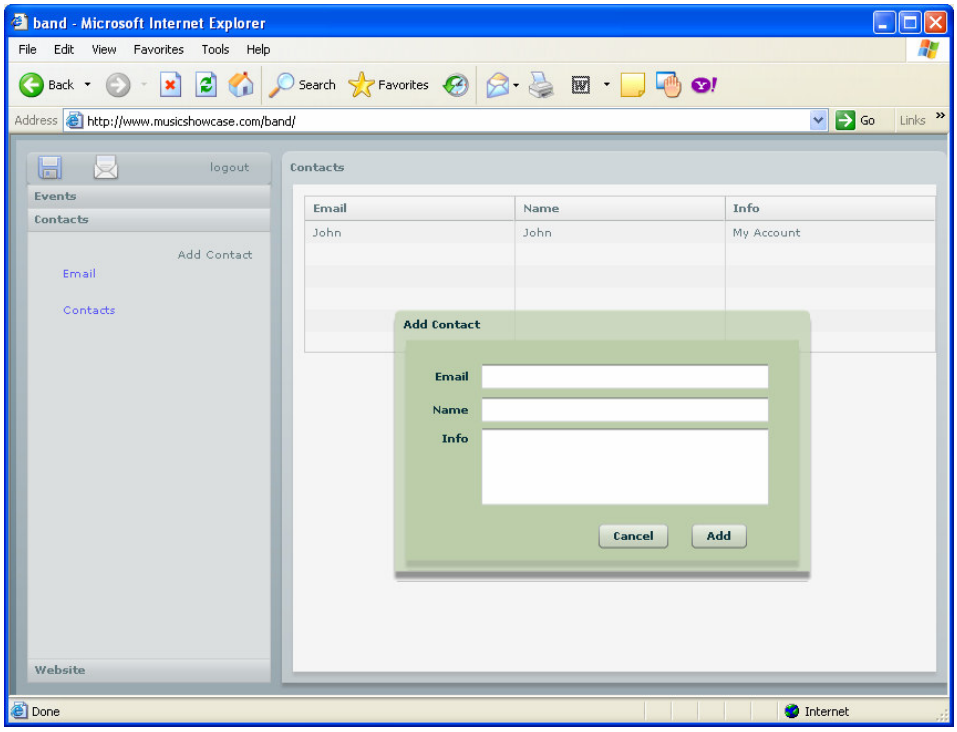


Figure 4.5.5: Add a Contact Screen.


ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).

# 5 Testing

## 5.1 Functional Testing

All testing was done during development. Rails encourages incremental unit testing concurrent with writing the application code. It also has a somewhat strict MVC architecture. The automated testing available in Rails isn't yet fully integrated into the web service. However, prior to writing the web service, I found it helpful to start by writing a traditional application with a minimal set of functions. Instead, all code can be tested from a page that links to the service. The coding is done in the development environment that gives feedback on the results. Within this environment a separate database is used. The program can then be made live with the flick of a switch.

Here is a description of the development / testing cycle. In Rails, you test each part as you go. There is no need to compile or reboot the server unless a change is made to the database structure. For this reason, I started by carefully planning out the database structure. When you encounter a bug, you fix it immediately before moving on. That minimizes final bugs in the core data of the program.

The user interface required a similar build / test cycle. It compiled on my machine in a few seconds. Then I would start testing and making adjustments.

## 5.2 Test Results

There are two core problems identified with this program so far. The first results from the fact that I had to implement the modal windows for editing with my own custom code (modal windows are not fully implemented in Flex Beta 2). I occasionally have had a condition that caused the program to be modal when it should not be. This was the result of receiving incorrect data from the server. The incorrect data from the server was fixed and the bug has not appeared again since. The second is that even though I ensure that a user is signed on and only interacts with their authorized data, I have not yet implemented any handling of errors on the server side. What basically happens is that if the service returns NULL, which it did only during development, a big message is displayed in Flash. Again, this only happened during development and has since been fixed. Basically, I covered all the standard cases for data passed between client and server. You will also notice some checks are done on the server side to ensure that the data is correct; if not, it will pass back a message to the client. This needed to be done on the server side to cover all future clients. However, for increased security, the data should be validated as much as possible on the client side, too.

# 6 Conclusion

This software project will be hosted on the domain http://www.musicshowcase.com for musicians that post their music there [1]. It uses Adobe Flash version 9 for the client-side. It was a very fun and challenging project to work on. I learned Flex [4] and Ruby [3] this semester but I already had a good foundation on the remainder of topics through my education at Rivier College.

## 6.1 Future Directions

I plan to continue this project. My current plans are to continue building the core of the site and then integrate this application with the rest of the application hosted at http://www.musicshowcase.com/ [1].



ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).

## 7 References

[1] Dion, J. A. Project Main Website. Retrieved November 4, 2006, from http://www.musicshowcase.com/

[2] MySQL. Retrieved November 4, 2006, from http://www.mysql.org/

[3] Ruby on Rails. Retrieved November 4, 2006, from http://www.rubyonrails.org

[4] Flex Beta 2. Retrieved November 4, 2006, from http://labs.macromedia.com/

[5] AJAX: A New Approach to Web Applications. Retrieved November 4, 2006, from http://www.adaptivepath.com/publications/essays/archives/000385.php

## Glossary

RAILS – Ruby-on-Rails is a way of programming that enforces MVC from the start [3]. At the core are many modules that allow for speed of development. CRUD and ORM are an integral part of Rails.

CRUD – Acronym for the core functions of the database. Create, Retrieve (read), Update, and Delete. Rails focuses around CRUD and ORM if you use the scaffolding (highly recommended for learning Ruby) as a starting point.

ORM – Object -relational mapping. In Rails, most objects directly map to the database; e.g., creating a new user maps directly to the 'users' table.

MVC – Model-view-controller. A style of software design that keeps a clear separation between the core components of a system. Simply put, the *model* is the logic for the data exchange. The *view* is generally the user interface. The controller handles the events in the system.

MySQL – A popular database that is available at most developer focused web hosts [2].

SOAP / RPC – A specific type of remote procedure call. Used to stand for simple object access protocol. But now it is just referred to as SOAP. Not the *cleanest* in that it has a lot of overhead compared to other RPCs, but well supported.

AJAX – Asynchronous JavaScript and XML. A new style of programming JavaScript on the web that does not require refreshes of the entire page [5]. Only the components in a page that you want updated will refresh; e.g., Google™ maps and Amazon's A9.com search engine.

FLEX – An XML style of coding Flash components [4]. The code is written in MXML and ActionScript. After compilation, you have a Flash SWF file that can be used on the web.

---

* **JOHN DION** received his M.S. in Computer Science from Rivier College in May 2006 and B.A. in Psychology from the University of West Florida in Pensacola, Florida in 1996. His current research interests are in Cognitive Science, Philosophy of Mind, Semantic Networks, and Human Computer Interaction for users with disabilities. His personal interests include contemporary Christian songwriting and homelessness prevention. He is currently working as a contract programmer and web application developer. He can be contacted at jdion@musicshowcase.com.



ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).