Alberta Historical Cemeteries:
Project Final Report

Course:	CMPT 498 (Team Capstone Project)
Team:	Michael Hnatiw
	Patrick Sawyer-Bennett
Start Date:	01/09/2013
End Date:	16/12/2013
Institution:	MacEwan University
Supervisors:	Dr. Indratmo
	Dr. Boers
Product Owner:	Dr. McKenzie
Contributors:	Canada Headstones Project
	Alberta Genealogical Society
	Service Alberta
	Archaeological Survey of Alberta
	Alberta Family Histories Society
	CanadaGenWeb Cemeteries Project

Table of Contents

Summary

The Alberta Historical Cemeteries Project is about providing easier and more complete access to cemetery records. The value of this project is that it provides researchers, the government, and the public with a means to better understand the culture of Alberta and preserve the records that may otherwise be lost. The main problems that exist currently are a lack of standardization in data collection and storage and no centralized means to access the information. Different groups collect data, which is most important to them, and do not communicate with each other in order to exchange data.

The core aim of the project was to create a foundational web application and database that could be expanded upon in the future to bring the project to completion. Notable achievements include implementation of searching for database records, a secure user sign in and authentication system, and a parsing system for existing records, updates, and exports. In this report we outline the project background and the initial goals that were defined in the proposal. We then describe the design that was decided on for each part, the implementation process, and the problems we encountered along the way. Finally, future development is discussed as well as resources for developers to become familiar with the technologies we used.

Background

As identified in Dr. McKenzie's Special Projects Grant Proposal 2013, the issue is that there is a clear lack of coordination and infrastructure in place to maintain and track Alberta's registered and unregistered cemeteries. The project aims to provide a web application for easier access of information, a centralized database for information storage, as well as a standardized method for data collection. Dr. McKenzie also hopes to bring together the various groups with cemetery data collections and encourage them to integrate their records into the project.

In order to complete the computer science degree at MacEwan University, students must complete a capstone project. With that in mind, we approached Dr. Indratmo to request that he act as a supervisor on a final project. Dr. Indratmo encouraged us to take on the Cemetery Project, which he had become involved in already. We formulated a project proposal document and received approval to work on the project beginning in September of 2013.

Aims and Objectives

The project was broken up into two-week sections with deliverables and milestones identified in each part. Also, the team and supervisors met every week to discuss progress and identify any problems that had arisen. The project was broken into three main tasks. The first was the creation of a relational database model to store

the source data. The second was the design and implementation of a web application to allow users to access, add, remove, and modify the data. The third task involved extensive further development of the web application to include as many of the features that a production deployment would have as would be feasible in the amount of time available. The project is intended to provide a foundation for further development since the time period needed to fully implement and deploy an application of this size and complexity is more than was available.

Design

Database Design

The design of the database involved deciding which relational database management system (RDBMS) to use and after considering the options, we decided to use the MySQL Community Server version since it is freely available and provides all of the features needed and has a record of reliability and wide usage. MySQL provides straightforward interaction through the queries provided and initial implementation of a test database was a simple process. It also provides support for the future of the application development as it scales up to support massive data support. Performance was another consideration when we chose MySQL since the production application will be constructing complex queries and running exports on the system of what could be potentially hundreds of thousands of records. MySQL is also highly customizable since it is an open source distribution.

Another early consideration was the discussion of whether our database would need to query other group's repositories each time a user wishes to access those records or whether we could provide all of the data contained within our repository. The performance considerations made it highly preferable that we contain all data to be accessed within our system and after presenting to the different groups at a workshop, it was agreed that this was the best method. The database design was based off of Dr. McKenzie's survey recording forms, which provide a very detailed method for collecting cemetery information. This also meant that the database had to be very complex to maintain the relationships. To make the design process easier, MySQL Workbench was used. This tool lets the developer graphically design the database and forward engineer the final product. This makes database development, modification, and testing much easier and speeds up the design cycle.

Web Application Design

Development Framework:

As outlined in our requirements document, the design of the web application was dependent on providing both relatively fast and efficient queries between the central repository and the web interface, and on providing users an intuitive and easy to use website that will allow them to view, submit, and interact with the

historical cemetery records provided by the source organizations. With these design goals in mind, a web framework to base code development on was chosen that would have to satisfy these needs, as well as the needs of future web developers (i.e. easy to learn framework, reduction in code complexity, MVC oriented separate compilation, testing modules, etc.) and to allow for expansion of implemented features as needed in the future.

Several frameworks were considered for the code base that provide differing functionality depending on what features need to be implemented, how you would implement them, and what kind of client-server interaction will be carried out by the end-users. The final decision for the backend framework to base web development on was **Ruby on Rails.** This is an MVC and Test Driven web application framework that uses the **Ruby** coding language to allow for simplistic code development through abstraction (i.e. simplifies code to allow for faster development of finished and functional websites). It uses built in methods and easily installable modules (called "Gems") to create webpage interfaces, add functionality for data processing, and it supports client-server communication and data collection to work with an integrated database. This framework was chosen above others considered because it seems suited to both our needs as developers (i.e. time constraints, current skills and knowledge, simplified code, easily learnable, etc.) and the needs of the project as a whole (i.e. efficient repository/web interface interaction, diverse functionality, etc). As developers with less than 4 months to work on all components of this project, it was imperative that we pick a framework with well-documented feature implementation to support development for ourselves and for future developers. Ruby on Rails provides extensive documentation, a large community base, and a wealth of online tutorials with which to learn from. While a challenge at first, learning the ruby on rails framework was enjoyable as it was apparent that you could do so much computation with little code if you understood basic ruby as well as the rails syntax and semantics. As well, Ruby on Rails supports a large range of descriptive methods that can be used to easily build DOM elements with set properties in order to make the actual HTML and CSS development easier for web site design. This was useful as neither of us has extensive experience with web design, so an intuitive way to create elements through method to cut down on the amount of work was desirable. One of the most important considerations for us was choosing something that would simplify database interactions, and Ruby on Rails provides that by implicitly querying a configured database through object methods that were easy to understand and use. This not only simplifies the code, but it gives users the ability to easily interact with the repository through the web interface, thus the ability to submit and retrieve records as needed was simpler to implement. Modeling database relationships is also supported and is very helpful as it allowed us to configure how our application would treat database tables in order to use their methods as normal and have these associations implicit in the code.

While the described advantages of Ruby on Rails helped ease development in some areas (i.e. less code) and allowed us to produce a functioning site in less time than if we used another framework with more complex syntax, semantics, and structure given no web development experience and our time constraints, it still had

its fair share of headaches. First, Ruby syntax was unfamailiar and a lot of little mistakes were produced that took a large amount of time to debug (i.e. colon on wrong side of word to create a literal string) due to human error. Second, a lot of the database querying was very implicit and was difficult to figure out how to extend from a code point of view (i.e. relate multiple models with differing relationships), as data relationships specified in models configured how the queries were carried out and sometimes caused problems (ie INNER joins ignored results with no foreign key values). As well, ensuring data integrity was preserved and that basic MySQL injection prevention measures were implemented took a bit of time to figure out as Ruby on Rails has limited measures to implement application security (only recommendations for coding style and what methods to use). Lastly, while documentation was extensive for their built in methods, I found it hard to extend the functionality of those methods to create custom DOM elements and custom functionality (i.e. input with search option menu) as most examples provided just displayed how to use them for basic implementations of the element. Thus, in order to figure out how to implement a lot of the search features, I had to turn to forums and rely more on the users of rails rather than the documentation. In short, while code is more simplified, the implicit nature of a lot of the functionality makes debugging and figuring out how to customize the given methods and modules difficult from a learn ability standpoint as new users of the framework.

Front-End Modules:

The Twitter Bootstrap framework was integrated into the project to provide a modern feel to the web interface for end-users. While our views about this framework were somewhat mixed at the start as its heavy use of the LESS style language was undesirable, after some debate, and given the usefulness of the bootstrap components, it was decided to integrate these components into the layout of the website. Their fluid containers provide an easily scalable layout and organizes content containers on the website. As well, their form and text components proved useful as their look was simple and provide functionality like a production website. Having little to no experience with webpage design, producing a layout from scratch proved more time consuming and did not function the way we wanted it to, thus I switched to using bootstrap components mid development and decided on a design that works with the general page layouts proposed in the web-mock-up's. As well, the simple look of the web components affords the end-users the ability for ease of use when interacting with the website (as we think the layout clearly states the functions of the web applications), which was an intended usability goal.

Implementation

Database Implementation:

For the implementation of the database, MySQL Workbench was used. Workbench provides a GUI interface for easy creation of tables and relations through the EER

diagraming tool. The script that generates the database itself can also be forward and reverse engineered using workbench. After going through the survey forms provided by Dr. McKenzie, the resulting design contained 46 tables for describing survey data. Figure 1 shows the top-level tables. The other EER diagrams detailing the sub-tables for the burial and cemetery descriptions are not included in this document but are available on the project repository. Implementing the database also required forming all of the relationships present between burials and monuments, monuments and cemeteries, all records and users, etc. These relationships are of utmost importance to the system in order to maintain data integrity and be able to make sense of the huge amount of data that will eventually fill the database.

User Accounts Implementation:

The implementation of user accounts became a focus during the final stages of the project. Having a fully functional account system was imperative to this project, since it relies on community involvement to accumulate records. In the design phase, we identified five user types that would access the site. We had public users who were not signed in, public users who were signed in, trusted users, researchers, and admins. To implement the feature, we first created a user model that contained information including the user's first name, last name, email, and password. We needed to control for valid input into the fields, so regular expressions were formulated to control email formats and ensure quality passwords. Also, the password, for security reasons, had to be stored in an encrypted form. We used the standard SHA64 encryption for passwords. To further the features of the user system, we implemented an email confirmation system. When a user creates an account, an email is sent to their address by the MacEwan mail server. The user must follow a URL in the email to confirm their account email before they may login to the site. Another feature implemented that is commonly seen on webpages was one to reset a password if the user forgets theirs. Similarly to the email confirmation, the forgotten password feature, when activated, sends out an email providing a link to a page to define a new password. To prevent a user from guessing the URL for password modifications and email verifications, a random token is generated using the SHA64 to identify the user and inserted into the URL. For password resets, the token is only valid for two hours from the time of creation. For the implementation of the foundation for the project we implemented a public signed in user and an admin user. We provide the admin user with the added features of being able to view all other users on the system as well as the ability to delete non-admin users.

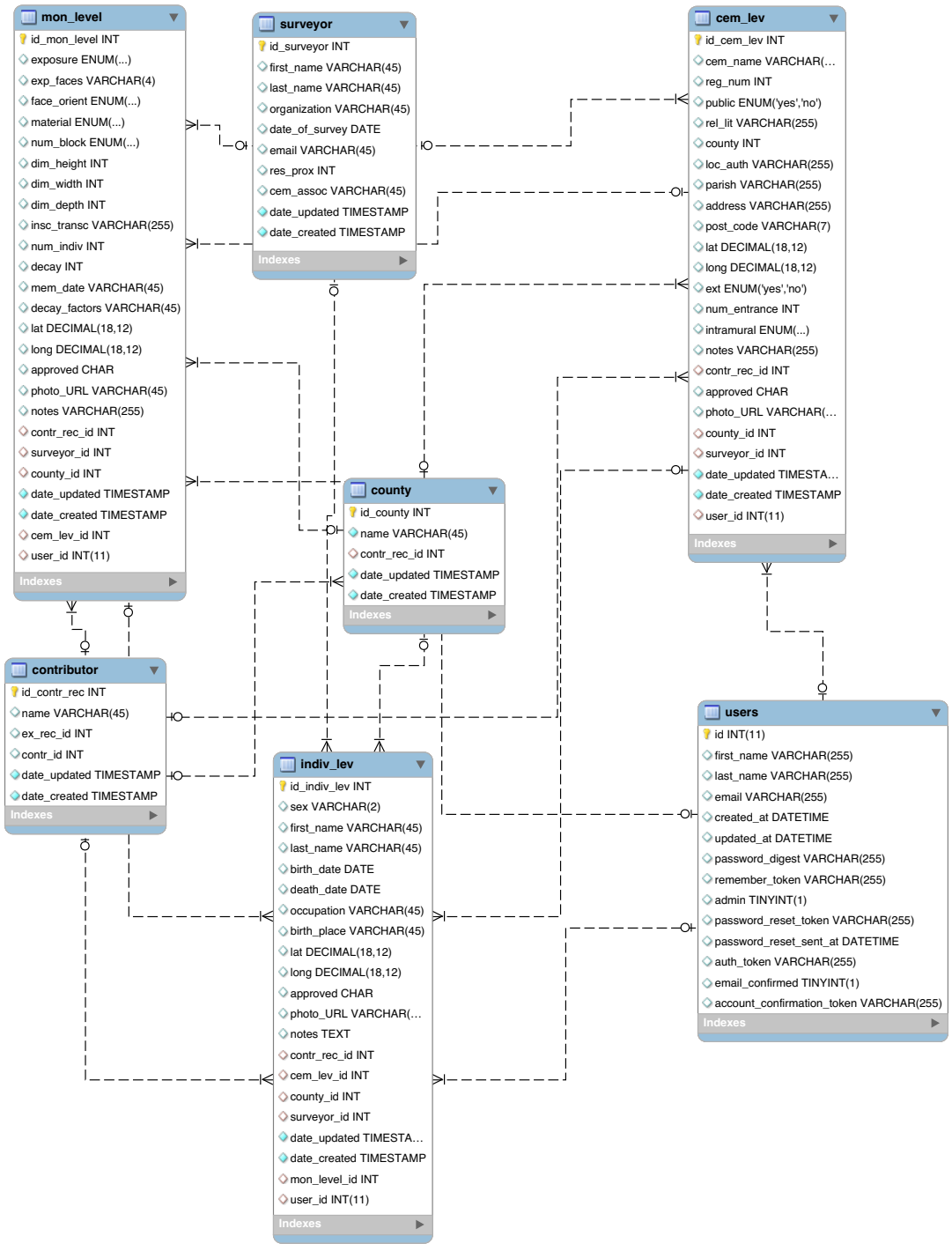


Figure 1: The main EER diagram shows the primary tables in the database. Sub-tables of the mon_level and the cem_lev are not shown.

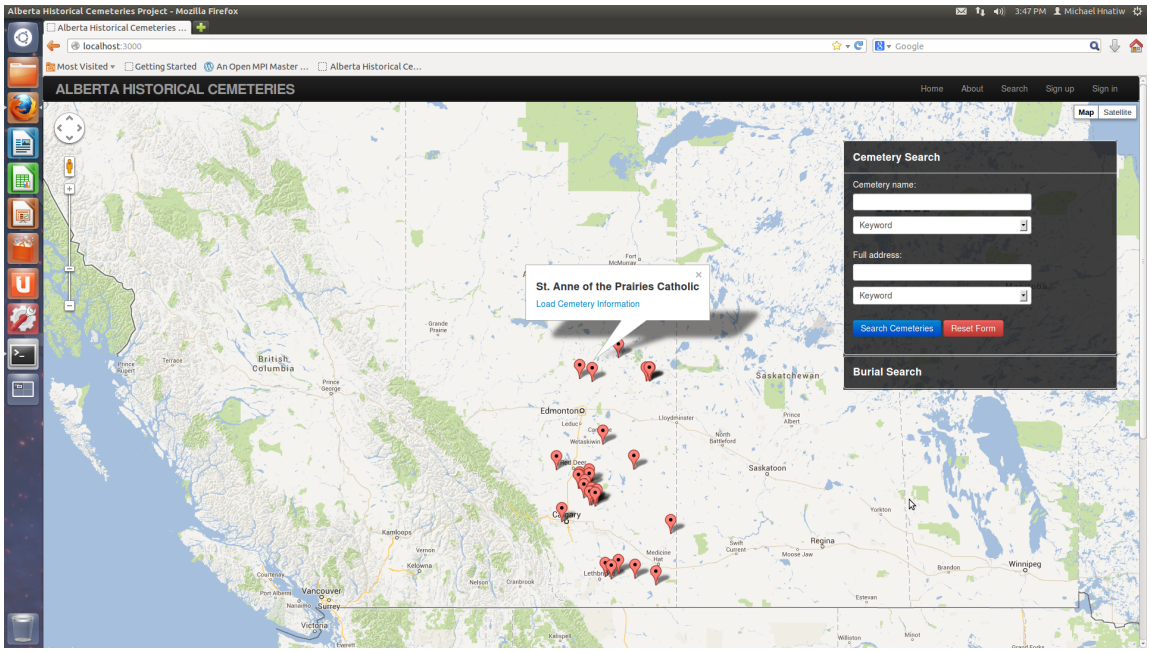
Web Application Implementation:

The web application is intended to afford the end-users the ability to easily retrieve and submit/contribute historical cemetery information, with data visualizations being key for users to experience more of an in-depth look into the data (i.e. infer patterns from burial locality and times of death on map), creating a detailed, intuitive, and helpful viewing experience for end-users. While all the features of the site were impossible to implement fully for a production environment given the time constraints, the basic search functionality of the website has been fully implemented given the amount and type of data we received so far from contributing organizations. As well, basic visualizations have been implemented using the provided Ruby on Rails Google Maps API (gmaps4rails) that renders cemetery pinpoints on a main map, as well as individual cemetery maps on each cemetery information page. Multimedia information such as burial and monument photos have also been implemented, but given the fact that a lot of the source data needs to be parsed and combined to generate absolute links to render these photos, in almost all cases links are just provided to the source organizations websites on the burial and monument pages. Lastly, basic user account functionality has also been implemented as described above, giving users the ability to create user accounts and sign in to view cemetery records restricted to users with accounts. To verify an account after its creation, an email is sent to the specified email address through MacEwan Universities SMTP client, emulating production account functionality. Administrative users (set on the server-side, not through the client) are also able to see all user account, and may delete them if possible (no submitted data important to the user).

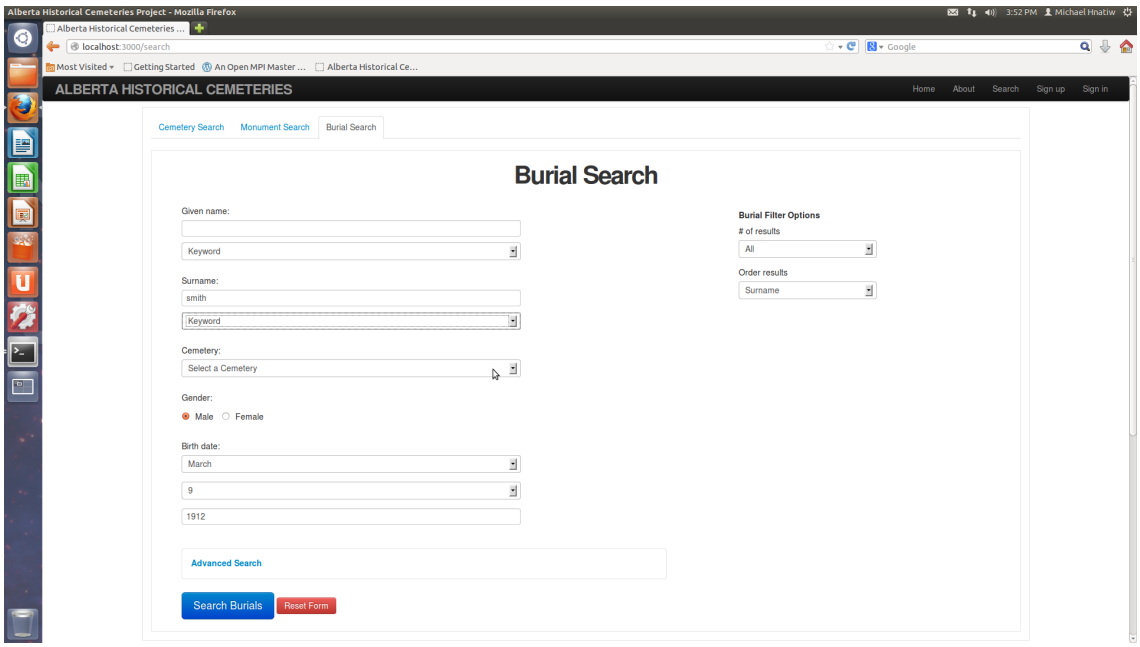
While the basic foundation for the web application and functionality are now implemented, there are still some planned features that fell outside our time range for this project. The submit forms feature that will allow users to submit cemetery data is functional for burial and cemetery records (for one or two fields), although very basic, still provides a proof of concept for how the submit form feature can be implemented given the full amount of data types that will be eventually integrated into the central repository. The visualizations are important for the end-users, but given that the scope of the visualizations were outside of the projected development time for this semester, not much has been done in the way of implementing them. However, the "gmaps4rails gem" does support polylines, overlays, and custom visualizations to be displayed over the map and interacted with by users (a helpful tutorial is provided in the Resources section). Another feature that was proposed but was not implemented was the extensive functions of the users dashboard page. Originally the account interfaces were specific to each type of user, giving an end-user the abilities to view submitted data, export data, delete data, or manage other user accounts given their account type. Not all user types were created, but for a proof of concept a general "public user" account and an "administrator" account were created to depict to the difference in access to features based on the account type (i.e. Administrators can delete other user accounts). However, it would be easy to implement other user types simply by setting a field within the central repository.

If given more time, each of these features would have be delved into in a more in-depth in order to provide end-users all the functionality ambitiously proposed for this development span.

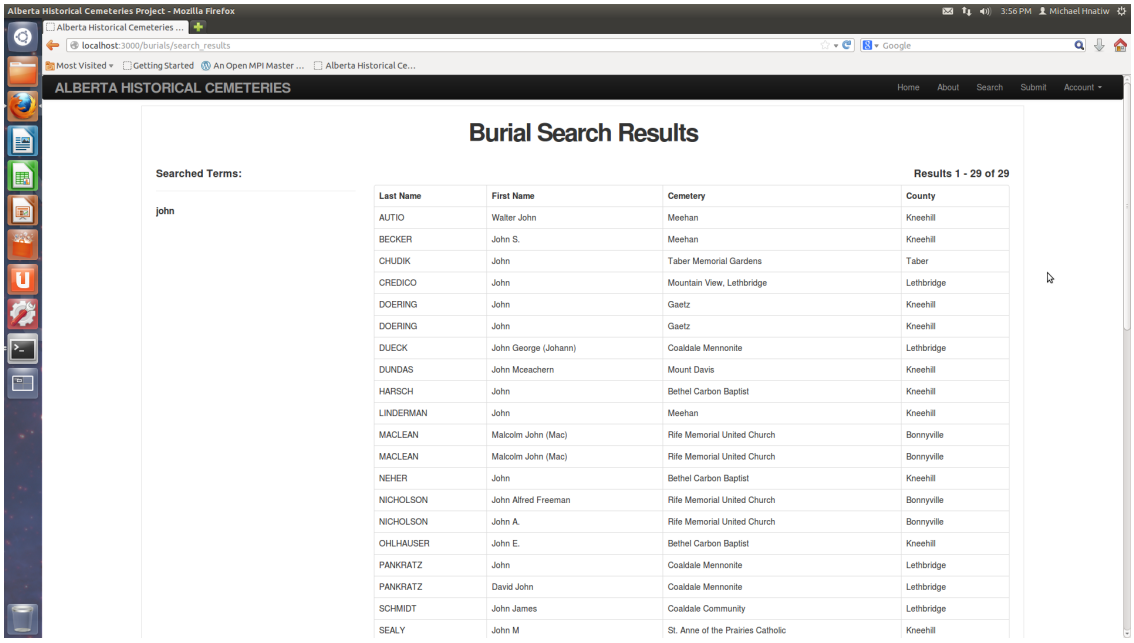
Screenshots



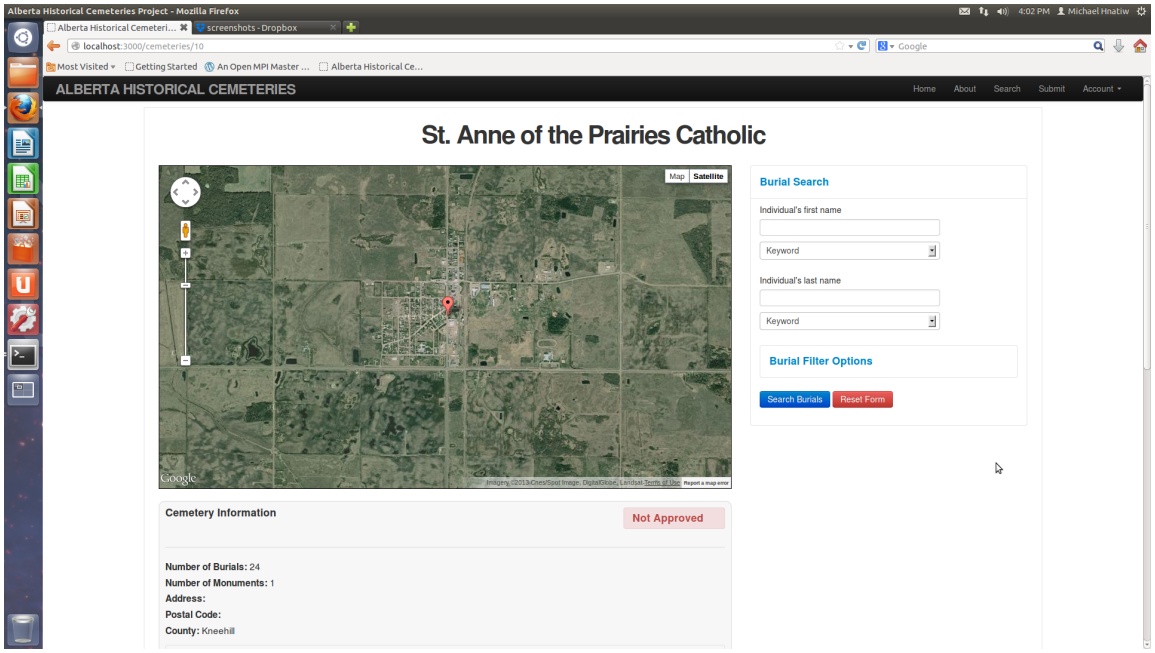
Screen 1: The main over map displaying cemetery pin-points within Alberta. Main map takes the full page (footer down). Search container for basic searching for cemetery and burial records.



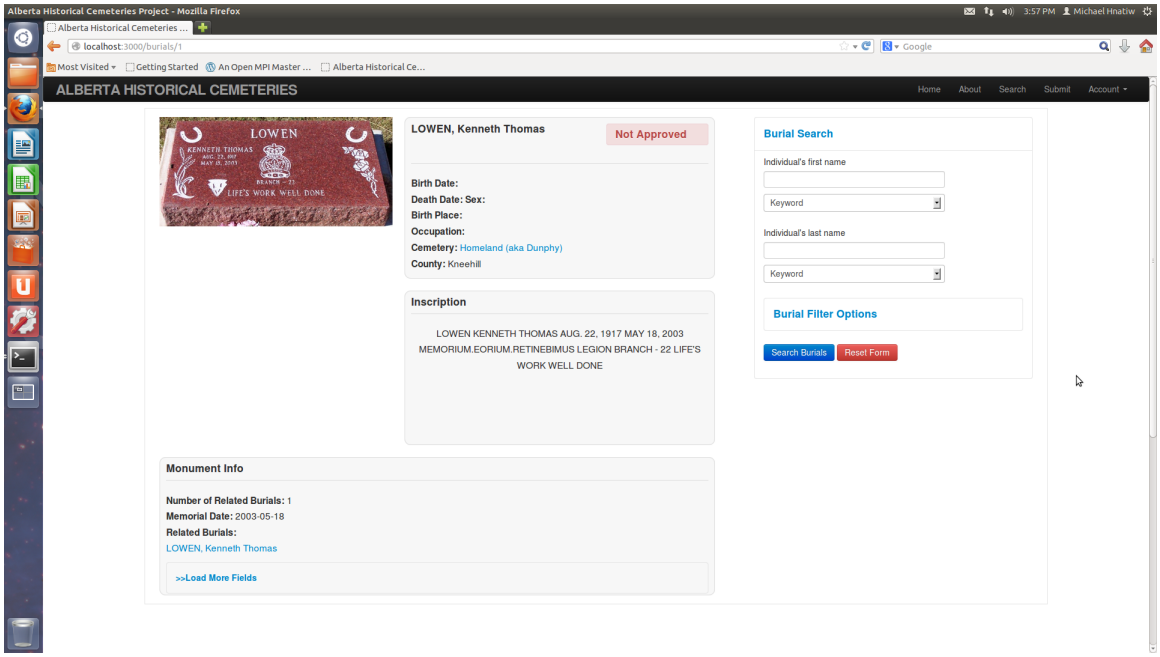
Screen 2: The tabbed delimited search page containing extensive search fields for burial, monument, and cemetery records. Filter options and advanced options provided along with combinations of form elements (ie date fields, radio buttons, etc.).



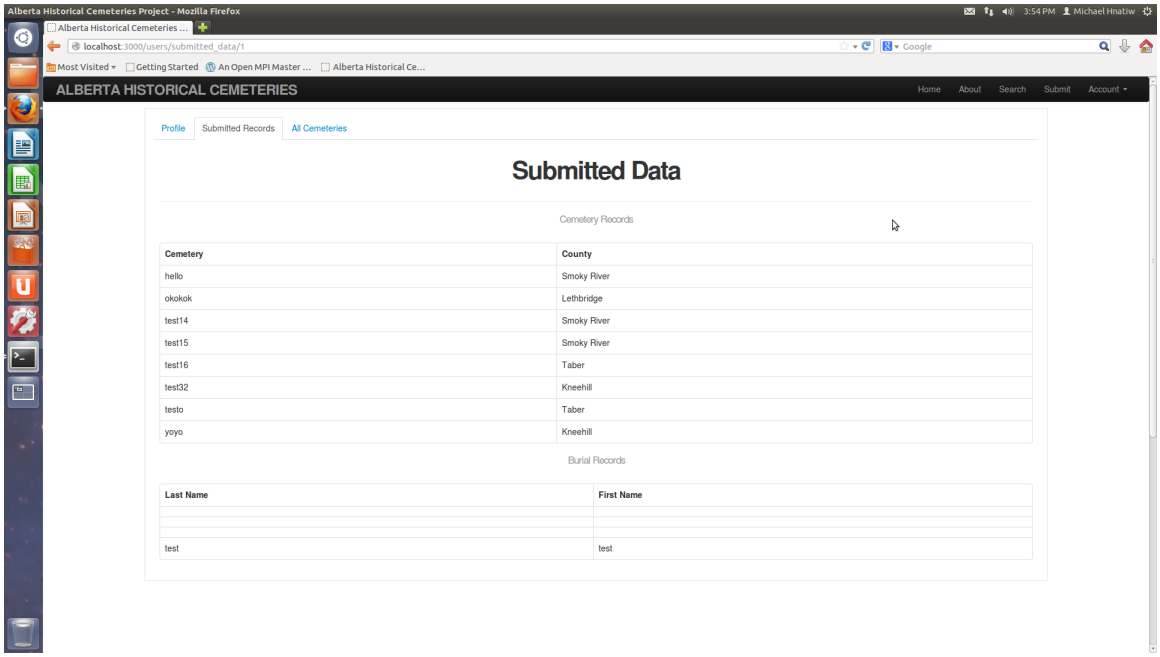
Screen 3: The table results page displaying queried burial, cemetery, or monument records. Each row links to corresponding record page. Added feature of listed search terms in left as well as space for other features or containers (ie advertisements, category listing, etc.)



Screen 4: Individual Cemetery information page. Contains a map interface displaying cemetery (will be used for visualizations). Contains various information containers contain record data (including related burial and monument lists not in screenshot). Sidebar used for searching for burial records within the cemetery (can expand in the future).



Screen 5: Individual Burial page containing photo of burial. Includes various information containers for burial and related information. Sidebar implemented to produce a new search for another burial record (can be expanded on in the future).



Screen 6: Tabbed user account dashboard display features available to current user (ie can view previously submitted data). Ideally will restrict base don user account type.

Recommendations

There are many features that will need to be implemented in order to bring the project to the desired production level specified by stakeholders and involved organizations.

Future Development:

-Comprehensive survey submission forms:
    - to emulate physical forms and their relations
-Addition of user types:
    -**trusted user**
    -**researcher**
-Refinement of CSS style sheets:
    -use of proper LESS syntax when overriding elements
    -better CSS style sheet implementation of classes, id's, and their properties
-Visualization Implementation:
    -polylines and overlays in Gmaps4Rails?
-User Account Dashboard:
    -proper implementation of twitter tabs or AJAX tabs (as of right now are links).
    -Export feature, proper profile settings, more customizability.
    -Researcher and Administrator specific dashboards.
-Integration of "**Sunspot Gem**" in order to use the Solr search engine for more efficient relational database indexing and querying of defined objects:
    - not implemented because general overhead of extra syntax and modules not worth it for smaller databases with less traffic (ie less than 1 million records), as well as have to pay for service.

References/Resources

There are many useful resources for learning about Ruby On Rails development and MySQL database management. The developers who take this project to the next level could focus on some of the mentioned resources to understand the project.

Screencasts of different rails development processes:
(proved to be very easy to learn from and useful for implementation of solid features)
http://railscasts.com/

For everything Ruby on Rails:
http://guides.rubyonrails.org/
http://apidock.com/rails

Main page for MySQL information and documentation:
http://dev.mysql.com/

Apache server project main page:
http://httpd.apache.org/

Main page for passenger, which works with Apache to host the application:
https://www.phusionpassenger.com/

For Gmaps4Rails extensions and documentation for visualization implementation:
https://github.com/apneadiving/Google-Maps-for-Rails
https://github.com/apneadiving/Google-Maps-for-Rails/wiki/Markers
http://larsgebhardt.de/using-google-maps-api-with-ruby-on-rails/ (polyline ex)

For Sunspot Implementation:
https://github.com/sunspot/sunspot/blob/master/README.md
http://www.sitepoint.com/flexible-searching-with-solr-and-sunspot/

Application Operation

Currently the application is running on a virtual machine on the MacEwan Students server. It uses the Apache server with Phusion Passenger.

Steps to operate

-From the command line, port forward to the students system and then to the project2 virtual machine

Template:
-ssh –L 88:localhost:<dest_port> <username>@students.cs.macewan.ca
-ssh –L <src_port>:localhost:88 <username>@project2.cs.macewan.ca

Example:
-ssh –L 88:localhost:1234 smithj@students.cs.macewan.ca
-ssh –L 1234:localhost:88 smithj@project2.cs.macewan.ca

-Open internet browser and navigate to localhost:88