Catalog: A Crowdsourced Solution to Geographic Cataloging

Ian Forsyth

CPSC 490 - Spring 2014
Advised By Avi Silberschatz
I. DESCRIPTION

*Catalog* is a combined web and mobile application that takes advantage of modern technology, habits, and behavior to give a user a clean and simple experience. The project implements two popular trends in computing – crowdsourcing and geographic tracking.

Crowdsourcing is the practice of obtaining needed services, ideas, or content by soliciting contributions from a large group of people, and especially from an online community, rather than from traditional employees or suppliers. With the popularity of today's internet and the intricate communities that have arisen from it, large problems can be solved on the smaller efforts of large groups of users. Crowdsourcing not only takes advantage of these communities but has more recently also been a key instigator in their creation. As technology becomes more and more mobile and accessible the potential grows.

Another shift happening along with the slide towards more mobile technology is geographic statistics. The ability to track a user's exact location, whether willingly given or constantly monitored, opens the door to many interesting metrics and analyses. Now a dynamic variable, where and how a user is interacting with the software can be inspected and conclusions about preferences, popularity, and market trends can be drawn.

Beyond including two growing higher level ideas within computer science, *Catalog* also spans several disciplines including web development, mobile app development, api programming, and database construction. Getting to work across different platforms offered a good challenge in variety and planning. The pieces were created in parallel to allow for testing and several refactorings made for a slimmer, better structured, more concise final product. The idea of *Catalog* in its entirety is beyond the scope of a CPSC 490 project. So, since the original proposal, this project set out to be a proof of concept and early implementation. That said, it was built with the broader final version in mind and now serves as a good launching-off point for further development and exploration.

To better understand the concept behind *Catalog*, consider a sample case of a Yale School of Forestry student wanting to catalog all the Elm trees in New Haven:

1. The user will enter a url and arrive at a web interface. He creates a profile with a name, email, password, and other personal contact information to be determined.
2. The user logs into the dashboard. From there he will have the option to view one of his catalogs, edit one of his catalogs, create a new catalog, or add an existing catalog created by another user. At this preliminary step, the dashboard will be empty so he opts to create one. There are three types of catalogs a user can create.
   1. Private – This catalog does not follow the crowdsource idea but is available for users trying to build their own data sets. Only the creator is able to contribute but they can decide whether the catalog is visible to the public or not.
   2. Contributor – For this catalog, the creator will invite other users by their usernames to join the catalog. The contributors will not be able to edit properties about the catalog itself, but they can log geographic points. The creator can decide if the catalog is visible to the public or not.
   3. Public – In this catalog, any user can contribute to and view it. However, it has a unique feature in that every geographical point has a confirmation rating. Users can give a positive or negative rating to each point, removing some of the potential for incorrect points.
3. After a catalog is created the user will exit the web interface. Using an iOS device, he will then...
use his username and password to log into a native application. He will see a list of catalogs he's created, has been invited to contribute to, or opted to contribute to (in the case of a public log). He will select the log he wants to add a point to.

4. In the case of the forestry student, the user will walk up to an Elm tree. The user will enter any information required by the catalog creator for a point (size, age, name, color, etc.) and then log the point. Along with the populated fields, the application will also submit the device's current longitude and latitude coordinates.

5. Once a point or collection of points has been logged, the user can return to the web interface to see a list of the points and the points laid out on a map for viewing.

By taking this idea, focusing on the important aspects and in turn toning down the scope, the final list of deliverables serve as a strong start. Built using a combination of PHP, Javascript, Objective-C, and MySQL and being supported by several key libraries, the final product of this project is exactly that.

II. DATABASE

The first step of the project was to create a database that could store and handle all the user and log information. Using MySQL, a relational database was created with the following tables and relations. The create statement for the tables can be found in the attached source: /lib/database.sql.
Clearly shown, the tables are closely related and all necessary to the overall project. They are as follows:

- **loggers** – This table stores rows containing an individual user's information. This includes their username to be used as their identity throughout the site, an ID that's used in all backend processes, and their personal details like name and email. This list of personal details could easily be expanded to include more. There is also the password column that contains a bcrypt encrypted password as well and an additional salt just for the sake of being thorough (expanded on in section V).

- **logs** – This table stores rows about the individual logs that users have created. Each log has a unique ID, information about what type of log it is (private or public), and other information such as the date it was created, a name, and description. Deleting a log starts a cascading delete that also removes all instances of it from the log_list and log_contributors table. The cascade continues to delete all points of that log too.

- **log_lists** – This table is simple but important. Each row contains a logger_id and log_id storing the logs that a user is currently contributing to. The user's “Log List” can consist of logs the user created, was invited to contribute to, or were public.

- **log_contributors** – This table also stores a logger_id and log_id and holds the information pertaining to permissions. If a user is invited to contribute to a log, it is added here. Without a row in this table, a user cannot contribute to a private log.

- **points** - This table is also crucial, it stores the actual points a user logs. Information such as the geographic coordinates, a name (implemented beyond the original proposal), which user logged it, and date created are used. And, because the database was designed with the broad concept in mind, it also contains fields for notes about a point, whether an image should be included with the point, and the reliability of a point for public logs.

### III. API AND DATA SERVICE

With the database in place, it is important to have a good way to communicate with it. The means to do so should also be independent of the applications so only one service is needed that can be used by both the mobile and web platforms. The data service CatalogData.php (attached in the source) is a class that includes a function for connecting to the database and a series of other aptly named functions that communicate with the database. They consist of program logic and one or more SQL statements – inserting, updating, and deleting information. Those functions are used throughout the project to populate the interface, create new logs and users, remove old unwanted ones, and edit existing data. This includes but is not limited to:

- `register()`
- `validateLogin()`
- `createLog()`
- `editLog()`
- `getLog()`
- `deleteLog()`
- `getAvailableLogs()`
- `validateContributor()`
- `addContributor()`
- `removeContributor()`
- `getDashboardData()`
- `getLogList()`
- `addLogToList()`
- `removeLogFromList()`
- `addPoint()`
Beyond that, a RESTful api was needed. For cleanliness in the web interface including a portrayal to the user that changes were happening immediately, jQuery ajax calls are used as a go between. The api is also how the mobile application verifies users, displays their log lists, and allows the user to log new points. The api end-points can be seen in the attached source.

IV. APPLICATION FLOW

Like with most applications, it's important to register users and give them an identity that can be used to own and contribute to logs. However, it's important that this registration process is not complicated or inhibiting. So, Catalog lands on a login page and offers a new user the option to register if they have not already done so.
After that, the user is directed to the dashboard page. Originally, the most prevalent information is the list of logs the user currently has active. There are the logs the user has created, the public logs the user has added to the log list, and the private logs the user is a contributor to.

<table>
<thead>
<tr>
<th>CATALOG</th>
<th>LOGOUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE A LOG</td>
<td></td>
</tr>
<tr>
<td>ADD AN EXISTING LOG</td>
<td></td>
</tr>
<tr>
<td>YOUR LOGS</td>
<td></td>
</tr>
<tr>
<td>Elm Trees in the Elm City</td>
<td>EDIT  DELETE</td>
</tr>
<tr>
<td>PRIVATE LOGS</td>
<td></td>
</tr>
<tr>
<td>Yale Residential Dining Halls</td>
<td>REMOVE</td>
</tr>
<tr>
<td>PUBLIC LOGS</td>
<td></td>
</tr>
<tr>
<td>New Haven Green Space</td>
<td>REMOVE</td>
</tr>
<tr>
<td>Train Steps Between New Haven and NYC</td>
<td>REMOVE</td>
</tr>
</tbody>
</table>

Then there are expandable regions including the ability to add logs. This list is populated with logs not already in the log list that are public logs the user hasn't created and logs the user has been invited to contribute to.

<table>
<thead>
<tr>
<th>CATALOG</th>
<th>LOGOUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE A LOG</td>
<td></td>
</tr>
<tr>
<td>ADD AN EXISTING LOG</td>
<td></td>
</tr>
<tr>
<td>Hydrology Class Sample Testing Areas</td>
<td>DETAILS  ADD</td>
</tr>
<tr>
<td>YOUR LOGS</td>
<td></td>
</tr>
<tr>
<td>Yale Residential Dining Halls</td>
<td>EDIT  DELETE</td>
</tr>
<tr>
<td>New Haven Green Space</td>
<td>EDIT  DELETE</td>
</tr>
<tr>
<td>Train Steps Between New Haven and NYC</td>
<td>EDIT  DELETE</td>
</tr>
<tr>
<td>PRIVATE LOGS</td>
<td></td>
</tr>
<tr>
<td>Elm Trees in the Elm City</td>
<td>REMOVE</td>
</tr>
<tr>
<td>PUBLIC LOGS</td>
<td></td>
</tr>
</tbody>
</table>
Another expandable region is the ability to create a new log. During creation, the user can choose between a public or private log. If it’s a private log, another region will appear giving the user the ability to add contributors. If added, the new log will automatically appear in the contributor's log list.

The last expandable region is the ability to edit logs that belong to the user. The user can change the name, description, and add or remove contributors.

When removing a contributor from a log, the user is given a choice to keep the points that user has logged or to remove them along with the contributor.
From the dashboard, the user can also decide to view a log by clicking on the log name. The log id is passed to the next page and it is populated with that log's information. The points are displayed on the left in a scrollable column and then placed over a map on the right. The mapping interface was a combination of a few libraries, attributed to in the credits.

Clicking on a point in the left column will automatically zoom to it in the map and clicking the point on the map will display a dialogue box with the point's name.
The points are a result of the mobile application. Although not polished and not serving as a mirror to the web application in terms of functionality, the iOS app serves its purpose. It starts with a login so the user can be identified and given their corresponding log list.

The user is then shown a screen to log a point. The user must enter a name for the point before tapping the Log This Point button and sending the name and coordinates to the api (and from there, the database for storage).
Beyond the typical time and effort that went into this project, there were some more technical points involved and decisions that had to be made to complete it.

One such decision was the way to store a user's password. Obviously, at this preliminary stage of the product, security is not the biggest worry. It is to serve as a proof of concept and the robustness of preventing malicious attacks can be focused on later in the project's development. But the most important security feature is allowing the user to log in using a secure password that won't ever be stored in its original form. So, an encryption function must be used. For a long time MD5 was the hashing algorithm of choice and it's still commonly used. However, in 1996 a flaw was found in its design and although it wasn't fatal, programmers have been moving away from it. One alternative, the algorithm used in this project, is bcrypt. It was designed by Niels Provos and David Mazieres in 1999. Two qualities that make bcrypt a good choice is it automatically incorporates a salt and can be run multiple times to increase the resistance to brute force techniques. Despite incorporating a salt, Catalog also adds a salt to each user's password for an extra layer of security. The salt is randomly generated by pulling 10 characters (allowing for repeats) from a list of 52 uppercase and lower case English alphabet letters. That salt is stored in the database and combined with the user's password when verifying credentials.

Another aspect of the application was error handling. Faced with often naïve users, the application had to account for accidental actions and incorrect input. This was done several different ways. For example, usernames for Catalog are unique. To keep this constraint, it is set in the database and if a new user tries to register with a username that has already been taken, the application catches an error on the attempted insert statement and displays that to the user. Adding contributors to a log was another place error handling came up. Contributors are added by their unique usernames and an error could arise three different ways. This includes: the creator trying to add themselves as a contributor, the creator trying to add a username that doesn't exist as a contributor, and the creator trying to add a username that is already a contributor. Handled through a combination of Javascript and the database, all these situations are prevented and displayed to the user. It was also important to update the database immediately when adding new contributors because it had to be checked that the creator wasn't adding two of the same in quick succession. Beyond those two more elaborate examples there were other small fixes like verifying username and password at login, confirming removal and deletion with prompts, and making sure all the fields were populated when creating or editing a log. Error handling was also incorporated into the iOS app for similar situations.

A final more interesting point of the project was actually getting the geographic coordinates for points when logging from the iOS application. Uploading the latitude and longitude coordinates required including the CoreLocation framework in my project. From there the process required a CLLocationManager object to pull the data in and a CLLocation object to store and manipulate it. Once the manager is instantiated its set to obtain the maximum accuracy and then its startUpdatingLocation() method is called. That fires the coordinates to a didUpdateToLocation() call-method where the location object is set, rounded to six decimal places (accuracy up to 10 meters with that precision), and sent in a post request to the application api.
VI. UTILITY AND FUTURE

This application has a variety of uses, ranging from personal to business to academic. For example a college student could be backpacking through Europe and want to pinpoint a location and an associated picture for every stop along the way. Or a local township might want to know the location and condition of downed trees and power lines after a severe storm. Or a scientist may want to keep track of the exact location along a river bed where he took water samples for a study. Beyond the purposes for data collection, just having the data available could be useful as well. New parents could use a catalog of public schools in the area when deciding where to move. A restaurant franchiser could find the locations of his competition to directly combat them. Someone may want to live vicariously through someone as they road trip across the United States. There is definitely a use for it and the amount of potential data is endless.

Overall, the project explores an interesting area. Crowdsourcing is becoming popular and, more importantly, easier now that everyone is connected with smartphones. Applications should start taking advantage of that. This will be particularly interesting considering the possible dynamic nature of data being collected.

VI. CREDITS

My project used several libraries as support. jQuery is a Javascript library that makes it easier to manipulate DOM elements. It was used heavily to to make AJAX requests to the API and to improve the user experience with animations and immediate updating. A combination of Leaflet.js and the mapbox api were used to create the map interface a log's points are shown on. The clustering, point dialogues, and zoom-to-point features were features that needed to be custom programmed though. Finally, because the version of PHP running on the server I used for testing was still 5.3, it did not have bcrypt built in like the latest versions. So I included a library built by Antnee that adds the functionality.

4. https://github.com/Antnee/phpPasswordHashingLib

VII. WORKS CITED