HELP Line: A Call Log Management System

A Capstone Report
presented to the faculty of the
School of Engineering and Applied Science
University of Virginia

by

Niti Paudyal

with

Jasmine Cha
Kevin Chan
Allen Huynh
Apurva Kasanagottu
Ben Lowman
Seamus Lowry

April 8, 2016

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

Signed: _____

Approved: _____ Date _____
Aaron Bloomfield, Department of Computer Science

HELP Line: A Call Log Management System

Allen Huynh
University of Virginia
abh3hu@virginia.edu

Apurva Kasanagottu
University of Virginia
ak4zk@virginia.edu

Ben Lowman
University of Virginia
brl2xx@virgnia.edu

Jasmine Cha
University of Virginia
jbc4gy@virginia.edu

Kevin Chan
University of Virginia
khc8gx@virginia.edu

Niti Paudyal
University of Virginia
np8bx@virginia.edu

Seamus Lowry
University of Virginia
spl6yc@virginia.edu

Abstract

HELP Line is an anonymous telephone hotline service intended to serve the University of Virginia (UVa) community. The telephone hotline maintains a call logging system in which volunteers log call information to better service callers. The previous logging system was unstable and difficult to use; this led to the development of a new HELP Line system. This system improved the productivity of volunteers through additional features such as searching, reporting, training mode, and tagging callers. It also enhanced old components of the system which include notifications, correct NetBadge authentication, and comments. This system was developed using Django 1.8.4, Python 3.5, and MySQL. Future work includes increased optimization of searching and reporting, an emphasis on User Interface and User Experience, and an online chat system to complement the telephone hotline.

1. Introduction

HELP Line is an anonymous telephone hotline service intended to serve the University of Virginia (UVa) community. HELP Line is affiliated with UVas volunteer center, Madison House, and is run by Program Directors. These program directors are are students with extensive volunteer experience with HELP Line. Program Directors help train and manage 100 HELP Line volunteers, who field over 500 calls per semester and talk to callers about issues including depression, assault, and stress from school. HELP Line volunteers then record these calls in a logging system. Currently, they use a Django-based system that allows features such as adding a caller, adding a call, merging callers, creating comments, and maintaining a log. This system contains the bare requirements for logging phone calls; however, the lack of certain capabilities lead to inefficiencies and inaccurate recording. While the non-profit has a current system, it is unstable and unreliable; therefore, it is difficult to estimate how long it takes to complete its tasks. Also, the lack of search capabilities and ability to disable volunteers creates issues of duplicate entries and allows inactive volunteers access to the system. Furthermore, due to the unreliable nature of the former system, staff were often forced to record calls in Google Documents. Program Directors then transfer the calls from the document to the system, which requires additional time.

Given these limitations, HELP Line was selected as an organization to participate in UVa CS 4970/4971, the Service Learning Practicum (SLP). The SLP is a year-long computer science capstone fulfilling course that focuses on agile software development model projects. The course is split into monthly phases that consist of two iterations, which are two weeks long. SLP pairs capstone groups with non-profit organizations to build a system for the organization and allow students to work on a project that affects real users.

2. Related Work

HELP Line previously used an older, custom, online logging system. This call log fulfills most of the requirements of our system. However, their old system is difficult to use, contains many bugs, and is very slow. There is one developer in charge of maintaining the old system but the server is frequently down, and HELP Line volunteers must resort to using Google documents to store call and caller data. Program Directors then input all the missing data once the website is functioning again. The old system also lacks features such as generating reports, notifying volunteers when others comment on their calls, and providing a training mode for new volunteers. In addition, the old HELP Line contained inaccurate and corrupted data due to mishandling the deletion of both volunteers and callers.

While there are several commercially available products available, HELP Line has unique requirements that prevent them from easily using such products. These requirements include leveraging UVas authentication system, managing complex merge operations, and a custom permissions structure. While possible to imitate on a commercial solution, such features are impossible to replicate.

3. System Architecture

Our system was implemented using the Django 1.8.4 web application framework on Python 3.5. Django is database-driven and follows the Model-View-Controller (MVC) paradigm. Most of our application is implemented within the confines of this framework. User-facing HTML (view) is rendered using the Django template engine, database schema (model) is specified using special Django derived classes, and the application logic (controller) is a set of Python functions callable by specified URLs. Django's object-relational mapper (ORM) manages the underlying database implementation.

The nature of the HELP Line system is that most useful operations are either querying tables from the database or adding new rows to existing tables. Consequently, the controller logic of the HELP Line system is relatively straightforward. Only a small portion of the application code is dedicated to implementing HELP Line specific algorithms. The following is a brief summary of the core components of the application:

- `views.py`: Contains application logic. Retrieves information from the database and serves information to the client via rendering HTML templates.
- `models.py`: Python classes defining the data schema. The Django ORM uses this information to create corresponding database tables.
- `forms.py`: A collection of classes that specify how the user may change model data in the database.
- templates/*: A collection of hierarchical HTML templates that specify the structure of each page in the web application.
- `urls.py`: Mapping from URL strings to controller functions in `views.py`.

While the majority of our application was developed strictly within the Django framework, it is worth noting that a significant portion steps outside of the Django paradigm. A large amount of JavaScript was required client-side to provide the dynamic user experience necessary for some pages. All pages that incorporate JavaScript extensively leverage JQuery. In addition, many also use Asynchronous JavaScript and XML (AJAX). While Django does not preclude the use of such technologies, their use is not directly supported.

3.1 Add and Edit Callers

When adding a caller, callers must be identified by a name chosen by the volunteer, an age range of either 0-18, 18-25, 26-40, 41-60, or 60+, characteristics, a gender, their story, determine whether they are a UVa student, and optional fields for call topics. The characteristics and story fields are markdown supported and thus can include a variety of styling. All volunteers can add callers through the add callers page; however, only Program Directors can edit and delete callers. When editing callers, the instance of that caller is repopulated into a form to allow for modifications on the caller details and overrides the original.

3.2 View Callers

The view callers page is the natural entry point for exploring all logged data in the HELP line system (Figure 1). This page is dual paned: the left pane displays a selectable list of all callers, and the right pane displays all logged calls associated with the current selected caller. In addition, the list of all callers can be instantly filtered and navigated with the keyboard.

As nearly all data in the database is indexed by this page, sending all data dependencies is not feasible. Instead, only the list of caller names is transferred to the client on initial page load. When a caller is selected, an AJAX request is made to the appropriate Django view. This view returns the rendered template containing all previous calls. The rendered template is inserted into the right pane using JQuery. In addition, Django is instructed to prefetch all data from the database on initial page load, speeding up the AJAX calls.

3.3 Merge Callers

Given the anonymity of callers, duplicate callers are often unintentionally created in the system. When a Program Directory discovers a duplicate caller, the two (or more) callers are merged. Every call associated with the secondary callers is moved over to the primary

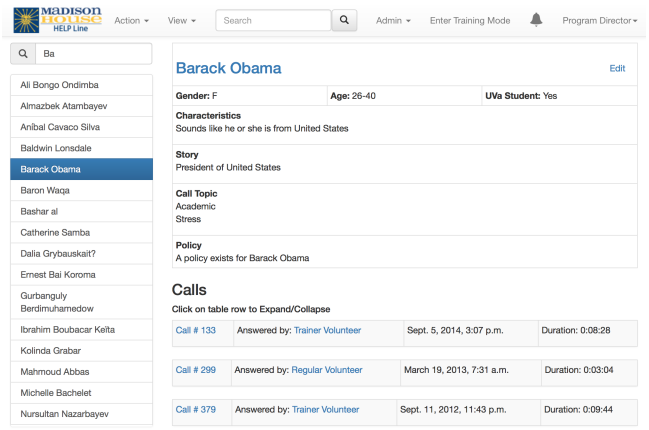

Figure 1. View callers page

caller. Additionally, the policies, characteristics, stories, and call topics of the secondary callers are appended to those of the primary caller. The primary caller retains its information about the callers name, gender, and age. The secondary callers are then deleted to finish merging. A caller cannot be merged with itself.

3.4 Add and Edit Calls

Adding calls is one of the most-used features of the HELP Lines system. Most of the work of volunteers is ingesting calls to HELP Line. Consequently, a large amount of time was spent optimizing its layout. It was discovered that volunteers would frequently browse the caller catalog (callers and their associated calls) while adding a call to more effectively identify repeat callers. To simplify this process, a dual pane layout was implemented containing both the call form and caller catalog. The left caller catalog pane can be quickly hidden with a button toggle.

The implementation of the caller catalog pane is nearly identical to that of the list callers page. The call form, however, has several notable features. First, the caller and volunteer selectors automatically update with type-ahead prediction. That is, the database is searched in real time for strings containing the typed substring. This feature is powered by AJAX calls to an endpoint established by a Django plugin. In addition, once a caller is selected, the caller policy for that caller is immediately displayed. The volunteer is given the ability to view previous alternatives suggested to the selected caller, and any number of alternatives may be added. The dynamic creation and deletion of alternative forms is accomplished using JQuery and Django Formsets.

Calls can be edited by the volunteer that took the call or Program Directors who have permission to edit and delete all calls in the system. Edit call is implemented simply by repopulating the instance of the call into a form and overriding the original. Deleting a call deletes all alternatives and comments associated with the call.

3.5 Move Calls

Given the magnitude of callers in the database, it is easy for Volunteers to associate calls with the incorrect caller. Program Directors can move calls they believe are associated with the incorrect caller to another caller. Comments are also moved to the correct caller.

In the move calls page, the Program Director first specifies the call(s) to move. The user can search for a call with the call id or the call summary, or both. The call to move field allows multiple calls to be specified. The Program Director specifies the destination caller, where the call will be moved, in the move to caller field. Since all

calls are associated with a caller, a call is moved by changing the foreign key relationship to the destination caller.

3.6 Alternatives

Alternatives are recommendations of the HELP Line staff to its callers. They are associated with a call, and are classified by an alternative type and a description of why the particular type is pertinent. The database table containing these alternatives cannot be accessed in the application directly; rows of this table are only visible when viewing the associated call. Alternative Types specify the general nature of the alternative, including if it is a referral to an outside agency. The set of Alternative Types is editable by Program Directors.

3.7 Call Topics

Call topics is an optional field for a caller. A call topic is a classification of an issue faced by the caller. These call topics allow volunteers to more precisely search the system for a caller or a group of callers. A caller has a many-to-many relation with call topics. In order to implement call topics as a many-to-many field, callers and call topics are two different entities in the database. Call topics is added as a many-to-many attribute to a caller.

3.8 Policy Updates

Policy updates is a feature that informs users of new caller policies or rules. A policy update is automatically created when a user edits the policy for a caller. In addition, a policy update can be created through the Admin navigation bar by Program Directors and requires information on which caller the policy applies to, the subject of the policy, and text explaining the policy. Only Program Directors can delete policy updates. Recent policy updates appear on the homepage when users first logs in, and all policy updates can be viewed by all volunteers from the view button on the navigation bar.

3.9 Hangups

HELP Line records all calls, even when the caller hangs up. Hangups are recorded with a click of button. The system records the time that the call occurred and the volunteer who recorded it. Hangups and calls are considered different entities on the database layer. Hangups can be deleted by Program Directors.

3.10 Comments

Program directors and volunteers can comment on calls to provide feedback and improve call quality. Adding, editing, and deleting comments occurs through AJAX calls to ensure the page is not refreshed. Comments can be threaded one level deep. Instead of using Django to automatically generate forms, they were implemented manually to allow more flexibility.

3.11 Notifications

Notification alerts volunteers when someone has commented either on a call they recorded or a comment they made. Program Directors frequently comment on calls to provide feedback to volunteers on what they did well or could have done better while taking a call. This feature allows volunteers to know when a relevant comment has been made and on which call or comment it was in response to.

A bell icon was added to the navigation bar. Every time a page is loaded, the number of new notifications for a volunteer is calculated from when a volunteer last clicked the notifications button and then is displayed in red on the bell icon. When the notification button is clicked, a bootstrap popover appears and displays a table with a link to a caller page for each notification. This table contains notifications for every comment for which the call or the parent of the comment is the volunteer logged in. After the notifications are viewed, the timestamp of last viewed notification is updated to show zero new notifications.

3.12 Search and Advanced Search

In the old HELP Line system, repetitive information was common due to a lack of search capabilities. Search queries the database to retrieve data associated with the provided keywords. The general search filters through all calls, callers, and volunteers with the given keyword. All relevant calls, callers, and volunteers are returned. For advanced search, volunteers may supply call summary keywords, caller attributes, the volunteer who recorded the call, date range, and any call topics associated with a caller. Advanced search only returns calls matching all the given keywords.The user may supply all six fields or as few as one. All volunteers are able to search. Searching is not logged by the system.

3.13 Training Mode

Training mode is a feature implemented to allow the customers to train new volunteers without exposing them to sensitive data in the production system. Trainee group members can still log in and become familiar with its use. Program Directors and Trainers control the members of this group. Only Program Directors and Trainers can switch between normal mode and training mode. Volunteers enter the system in normal mode, and Trainees enter in training mode. Callers, calls, hangups, and policy updates are not transferred between the two modes.

Initially, the design called for two separate databases, but implementation was too complex as much of the data would need to be shared across databases. In the end, each of the four models that would not transfer between modes were given a flag to indicate whether they were training data. In addition, volunteers were given a flag to indicate whether or not they were training data. When a request to read or write any of the four non-transferable models is received, only those whose training flag matched the volunteers training flag are presented.

3.14 Logs

The logs page is only shown to Program Directors. A log is created for almost all actions in the system and contains information about what action occurred, the date and time of the action, and the volunteer associated with the action. A helper method is called at the end of almost every controller method to create the necessary log object. A Program Director can regularly click a button on the logs page to clear the logs and delete unnecessary log data.

3.15 Reports

Reporting was the largest new feature requested by the customer. Reports on calls, callers, and call topics can be generated by Program Directors through the admin navigation. A CSV is generated based off parameters such as year, calls, callers, or call topics. Reports on calls can be generated based on the day, hour and day of the week of when the call was taken, and reports for callers can be generated based off characteristics such as gender, age, and based off the number of UVa students. Reports for call topics can be generated based off the number of calls with a particular call topic, and the percentage of callers to call topics. A bar chart is then rendered off the CSV utilizing the Data-Driven Documents (D3) javascript framework, and a frequency tip library that indicates the value of each bar on the chart. All data on calls, callers, and call call topics can also be exported as CSV for further analysis.

3.16 User Management

Program Directors can create a user by using a form to input information: first name, last name, email, username, password, group, gender, and graduation year. The email field is important

as the system can send an email to the user with the password using the SMTP information specified in the settings file.

Program Directors also have access to a page which lists all users. Due to the number of users, the page loads this data via AJAX. In addition, Program Directors can filter, edit inline, and delete the users listed on this page. Bulk operations are supported. When the Program Director submits the request to bulk edit users, an array of volunteer ids and the desired action is submitted to the server. Deleting a user also removes all content with a foreign key relationship to user such as the users calls, comments, logs, and hangups. As such, disabling a user is provided as an alternative which toggles a flag to determine access without deleting all associated content.

3.17 Netbadge Authentication

NetBadge is a service used by the University of Virginia (UVa) to authenticate on either a username/password combination or a digital certificate on the client machine. User information is sent to a remote server where it is authenticated. Then, the remote server sends back a response that puts the authenticated users computing id on the local servers REMOTE_USER variable. A computing id is a unique identifier given to a UVa student or employee for authentication purposes.

Django does not natively support this form of authentication. A link was added to the login page that redirects the user to an index.php file in the static folder. This file is protected by a .htaccess file that first redirects to NetBadge authentication routines. Once NetBadge has authenticated the user, index.php is called to place the validated computing id into the database along with a random fifty character string which is added for security purposes. Once that information is in the database, index.php returns control to a Django view, passing the random string as a URL parameter. In the Django view, the database is queried for a login token that matches the given string. Once found, it logs in the volunteer with the matching computing id.

4. Challenges and Design Decisions

4.1 Python 3 and Dropping Python 2 Support

The system originally was designed to only work on Python 2. However, after several months, it was found that the system would need to run on Python 3. One of the required changes included adding a try-except statements ensure version-specify imports did not cause crashes. In addition, the continuous integration test system was changed to use Python 3. It was eventually decided that Python 2 support would be dropped because it would allow the team to focus more on features rather than testing the system for both versions of Python. In addition, reporting on call topics uses Python 3 set functionality.

4.2 Data Transfer, Corruption, and Time Zones

After development was completed, the old data needed to be transferred to the new system. A SQL script parses the old systems tables and inserts the data correctly into the new systems tables. This system creates a one-to-one relationship between volunteers and users to leverage Djangos built-in Group and User functionality. The previous system did not, which means that appropriate user information was loaded into the auth_user table, including a copy of the hashed passwords, before the volunteer entries were created and associated. This allows volunteers to keep the same password they used for the previous site. Our system tracks more fields than the previous system and thus the transferred models will have NULL or default values for these fields (i.e. graduation_year for volunteers).

Another important data change is the permission schema. The old system used an access field assigned a number while the current system uses four Django Groups: Program Director, Trainer, Volunteer, and Trainee. Since the Trainer and Trainee permissions are new in the current system, no corresponding fields existed in the previous system. As such, members were added as either Volunteer or Program Director permissions. The old system had a table for call topics, the table did not appear populated as most of the entries were empty or unclear. As well, the client indicated that the system was not designed with this functionality, so this table was not merged over. Comments, hangups, and logs were transferred in a fairly straightforward manner.

Some corruption was discovered in calls table of the original database. Each call is associated with both a volunteer and a caller. The original system had 240 calls that were missing either their volunteer or caller foreign key id. The volunteer key was most likely lost by deleting them from the system and improperly cascading the delete, leaving no information about the original volunteer. It is unlikely that this information can be recovered. The caller key was most likely lost when two callers were incorrectly merged. Specifically, it is believed that these calls were copied and associated correctly, but the old call had a NULL primary key. Much of this corruption was solved because only 40 calls were missing their caller primary key. A dummy volunteer was created to be associated with the calls that were missing their volunteer primary key, and the 40 calls left without a caller were not transferred.

Due to issues in which Django could return or aggregate on only date components as the result of a filter, some MySQL code is specified explicitly inline. This code was used to return a table containing only the relevant date components for reporting, and, while it does limit database interoperability, no other solution was easily found. There may exist other database backends that work, but only MySQL is officially supported. Any transfer of this system risks breaking or violating one of these requirements through use of either a different Python version or different database management system. Specific instructions to transfer or install the system are given in the installation instructions.

5. Workflow

The purpose of the system is to aid volunteers while recording calls to HELP Line. There are four groups of users that will use the system: Volunteers, Program Directors, Trainers, and Trainees.

Volunteers frequently add calls. If the caller does not exist, the user fills out the add caller form. Volunteers can also look at past calls that they have been made. In the current Helpline system, the Volunteer can see their most recent calls and policy updates on the home page. In addition to checking most recent calls and policy updates, the Volunteer may want to find a specific caller. If a Volunteer cannot remember the exact name of the caller, the Volunteer can use the search bar to find keywords in the callers name or the callers story. The Volunteer can also search the callers name in the list callers page to view all of their previous calls. While viewing the previous calls, the Volunteer can edit their recorded calls and comment on any call.

Program Directors have similar workflow to Volunteers. Many of the PD actions affect the entire system. PDs have permission to add, edit, and delete alternative types, policy updates, call topics, and Volunteers. They are also able to merge two existing callers, move a call from one caller to another, view logs on the system, and generate reports.

The system also includes training mode. Training mode is a sandbox where new Volunteers, called Trainees, can practice logging calls and using the system. All Trainees automatically enter the system in training mode. PDs give existing Volunteers the permission to become a Trainer to mentor the Trainees. When a Volunteer becomes a Trainer, in addition to their existing functionality on the main system, the Trainer can enter training mode. When a Trainer

is in training mode, they have limited permissions, similar to a PD. In training mode, a Trainer can add a policy update, merge callers, and move calls.

6. Results

The new HELP Line system is an extended re-implementation of the previous system. The most important metric of comparison is the relative increase in stability of the new system. The old system, even during our own development, was only intermittently functional. Despite the short time our system has been in production, it is more thoroughly tested and has so far proven to be more stable than the prior system. No customer feedback was related to system instability.

The performance improvements of our system are difficult to quantify. When loading similar pages in both systems, the new system is marginally faster, though the non-scientific nature of such tests makes conclusions difficult to extrapolate. Our system includes several features that significantly speed up the work flow of volunteers. Small, productivity increasing features abound: automatically populating call fields after adding a caller, automatically notifying the user of policy updates, instantly filtering the list of callers, more advanced search features, comment notifications, and extensive use of type-ahead field population, etc. In our testing, the current HELP Line volunteers provided positive feedback regarding these features.

When conducting usability testing, volunteers, who possessed no prior knowledge of the system, were able to more quickly execute common work flows. The customer expressed appreciation for new features added such as searching, reporting, policy updates and split screen view when adding callers. In addition, our system produces statistics and exports raw data improving Madison Houses ability to gather analytics about HELP Line. This is likely the single largest functional improvement of the system.

7. Conclusion

We successfully developed an improved re-implementation of the HELP Line call log system. Our system, in addition to offering functional improvements, was faster in practical use due to a myriad of time-saving features. The HELP Line volunteers have expressed positive feedback, and have noted a definite improvement over the previous system.

8. Future Work

Future work includes improving and expanding newly added features like search and reporting. Reporting is limited to some number of fixed parameters for analysis. In general, future work could focus on database optimizations. While developing the current system, the performance ramifications of Django queries were not considered. The main focus of this project was on the implementation of functionality as dictated by the customer thus future work should have a bigger emphasis on the user interface and user experience. Bootstrap was utilized as the HTML, CSS, and JS framework which provided faster development but led to poor design decisions as the team relied entirely on this framework to handle both UX and UI. Lastly, an online chat system could be implemented.

9. Acknowledgements

We would like to thank Madison House, HELP Line, Rachel Winters and Sruthi Poduval for their support and help in the completion of this capstone project.