Greek Rush Management Application: Final Report

Josh Carroll: carrollk@seas.upenn.edu

Faculty Advisor: Susan Davidson

Abstract

Every year, fraternities and sororities in colleges all over the country host hundreds of freshman for rush. From this huge pool, the brothers and sisters must narrow down to a select few who are chosen as pledges. This process lasts a few weeks and involves a great deal of secrecy and information gathering. This ranges from the initial contact information to opinions and votes collected from brothers on potential pledges. Currently, this process is very time consuming and involves a great deal of paperwork, even on a campus with relatively small (30-40 members) Greek houses such as Penn. It certainly does not scale well on larger public universities with upwards of 200 members in a house.

A web application is an ideal solution for this problem. It can provide central data storage, as well as interfaces for viewing and altering it in various security domains. Users will be able to log on and view only the data that they have permissions for. The applicant pool can be populated through a variety of mediums, including online submission, uploading a spreadsheet, and possibly through the popular social network Facebook. Members will be able to voice their opinion on particular applicants anonymously, and admins will be able to request specific feedback from all members. This data will be viewable using various reports that can be filtered and customized by members. No data will be accessible to users without the proper security credentials.

Related Work

Surprisingly, there do not seem to be any products or systems that are publically available but provide a solution for this problem. I did find one product marketed to a similar audience but serving a different purpose. The “Fraternity Rush / Recruitment” application on Facebook\(^1\), is marketed to Greek Councils as a management and administrative tool for recruitment and rush. It allows officers to post information, calendars, and events that are viewable to the public, and allows rushees to apply and have their data be accessible to officers. This application is built on The GIN System\(^2\), for Group Interactive Network, provides communications solutions, primarily for Greek and other student organizations. This tool focuses more on the house communicating with pledges, while the system proposed here focuses on exchanging and storing the private data used by houses to make their decisions.
about pledges. However, there may be value to be gained by considering how this application works and looks, since its target audience is the same.

As to existing solutions, every fraternity or sorority has a different approach to solving this problem; however the two common approaches that I have seen at Penn involve either managing a spreadsheet through Excel, or reverting to an analog system (pencil and paper). Both of these approaches are difficult and time consuming to manage in terms of collecting and distributing data. The proposed system offers dramatic improvements. Additionally, it is more secure, and allows officers and administrators precise control over who can view what, as well as providing better mechanisms for anonymity. It can track multiple users inputting data simultaneously without concerns about version control. There is less concern of data being lost. Also, it will be easier to modify the views of the data.

Numerous web applications follow similar approaches to solving problems in other domains. Facebook handles data security through different levels of control over individual data and broader access based on groups (called networks). Other sites provide similar interfaces for polling, voting, and commenting on content, such as doapoll.com[3] and digg.com[4]. These sites provide functionality for anonymity but have less control over who can view and alter what content.

Existing research into several technologies and designs will provide some of the framework on which to build this system. In particular, Role-Based Access Control[5] will be used to provide the appropriate level of security control for this system. A Hierarchical RBAC as described in Sandhu, 2000[6] will be used. Under this model, roles with particular access privileges are defined, and roles with higher access are built from lower roles in a reverse-tree manner. Users are assigned a particular role, and their permissions are defined based on that role. User authentication will be provided through the https protocol via Facebook login, which implements two secure communications protocols, Secure Socket Layers and Transport Layer Security[7].

**Technical Approach**

This system provides two primary functions: to provide a web interface to the underlying database, and to integrate and supplement this database with data from the Facebook Platform. The database and interface will be discussed first. The database runs on MySQL, with the web application developed in Java Server Pages[8] and hosted using Apache Tomcat on a remote server provided by the web hosting provider hostjava.net, which also hosts the database. Primary development was done using the Eclipse IDE, and development for the JSP/html elements were done with WordPad.
This project initially planned to fully encrypt all communications using SSL, rather than just the authentication. However, it was determined that the benefit of this was not worth the cost. First, there is a precedent for this: Facebook itself does not currently encrypt all communications after authentication, so users are used to some of their data being snoop-able. Even if we felt we wanted better security, we would need to buy a signed SSL certificate, which is currently expensive relative to our user base. This may change as we move to commercialization. Alternatively, we could sign our own license for free, but this causes most browsers to complain. This approach is still susceptible to spoofing while creating an illusion of stronger security than actually exists.

The authenticated login procedure is handled by Facebook using their Facebook Connect\[9\] functionality, which provides users access to their Facebook account as well as much of the information and functionality of Facebook through a third-party site. This decision allows rushdB to outsource one of the most problematic and risky features (secure login and particularly password storage) to a professionally developed application. This is more pragmatic and less time-consuming than a homegrown solution, and is more trustworthy to users who already use this service. In addition, rushdB takes the additional precaution of re-authenticating with Facebook with each page-load to insure that a user cannot elevate to privileges they should not have. This is done by having the database record the role assigned to each user, and once the server knows which user requests a page, it can determine the authorization rights of that user.

RBAC is implemented using several hierarchical roles. Officers for a particular house can view and edit data on the current rush class and grant Member roles for their house to other users, and also use other features such as generating reports. Members can view contact information for all Rushees and can review Rushees anonymously. They may also have some limited access to viewing data input by other Members. A Rushee has access to view and edit contact information. This access is granted to any registered user. Members have all the
access rights of Rushees, and Officers have all the access rights of Members. All Officers and Members will be assigned to a particular house and only have access to the data for that house, and Rushees applying to that house. At the end of each rush season, the data from that year will be archived and accessible only by Officers.

The system is further segregated by University. Users declare a particular school upon account creation, and are entirely blocked from viewing any part of the database for other schools. This allows the application to expand to multiple campuses easily. Admin controls will eventually allow easy management of different universities, but currently this must be handled manually in the database. The system is also configured to give members of a house access to basic information on rushees at other houses in case they want to contact them. A future feature will include letting rushees opt into or out of this feature at account sign-up. This will not be exposed in any kind of aggregate fashion, and again, no information between universities will be available.

To join, the members of a house create user accounts and request Member status for their house. The Admin promotes an initial user to Officer for that house, and they handle promotion of other Members. They are logged a “pending” status in the database in the meanwhile, with Rushee access privileges. Officers do promotion through a special officer panel on the site accessible only to them. This provides a list of all users who have requested membership in their house and allows them to select the ones to promote. This is also where Officers access the reporting infrastructure, which allows them to export data through Excel.

Once the initial data on Rushees is in the database, Members are able to anonymously add and edit reviews on each Rushee. This includes numerical ratings and textual explanations on categories such as friendliness, dress, etc that can be specified by the Officers. All of this is managed in the database. Aggregate reports can be exported as Excel spreadsheets to be distributed by Officers. Officers and administrators have a separate interface to manage access and customize these reports. Data population methods have not yet been implemented beyond the Facebook account method. Members can look at different slices of rushees and members of their house through a list-style page that displays basic information, picture, and ratings for each user if applicable. This page currently supports sorting by last name as well as a keyword search over profile and comments for each user.

Search is implemented using an inverted index in the database that is updated as new data is submitted. Keywords are determined using a fixed set of stop words and Porter’s Stemming Algorithm\cite{10}, a standard algorithm for this procedure.

As mentioned above, the site is hosted on an Apache Tomcat server provided by a third-party hosting company, hostjava.net. Currently this Tomcat server is shared with a number of
other applications (because of cost considerations), which makes the platform less stable than it could be. There are plans to move this to a dedicated server at the point that incomes from this project make that feasible. Programming for this project was conducted on the programmer’s machine and then moved to the server via FTP. Administration of the server and database occurred through SSH and the Plesk administration panel provided by the host company. The database is accessed using a number of Java Servlets on the server that primarily do user authentication and data verification before storing data using JDBC.

Using Facebook Connect provides a number of benefits currently and provides the biggest avenue for future feature expansion. Sharing an account between both sites has been discussed above – no need to remember additional passwords, etc. The pictures of users are also provided by Facebook (using the default profile picture). This means that it stays current, and users never have to go through the trouble of uploading a photo. This is done using a Facebook-provided technology known as XFBML (eXtensible FaceBook Markup Language)[11]. Using this, the server leaves a tag in the JSP-generated page with a referred link. The client browser uses JavaScript to query Facebook for the image (a process known as Cross Domain Communication). The page renders with a blank picture frame, and once Facebook verifies the query as legitimate and sends the image back, JavaScript code re-renders the picture frame with the proper picture. rushdB also pulls user data from the Facebook Platform on account creation to reduce the time it takes a user to register. This is currently restricted to name as other contact information is not exposed by Facebook at this time. From the list view, we also query whether a user is friends with any user whose profile they look at, and provide a button to make a friend request if they are not. This is also handled through the Facebook Platform.

All Facebook interaction, including authentication and pulling data such as name or friend status, are implemented using an unofficial Java implementation of the library that lets sites talk to the Facebook Platform APIs. This library was downloaded from the Google Code database and is updated regularly. For authentication, the client caches Facebook-generated cookies for a given session and sends them to the server (or Facebook) with each call. The server uses this library to interpret the information and request that it be authenticated from Facebook. Facebook returns cookies which include the uid of the user who made the initial request. From this the SQL database on the rushdB server is polled for the role and house/university associated with that uid. The server then ensures that this role/house combination can view or store the requested data, and handles it accordingly or redirects to an error page. Communications with Facebook through this library are handled using the JavaScript Object Notation (JSON) interface[12], “a lightweight data-interchange format.” The library returns output in JSON as well, so the server has to handle unpacking and interpreting the data returned by Facebook.
For exporting the data from the database to an aggregate reporting interface, the Excel Spreadsheet format was chosen as a practical and widespread format for spreadsheets. To solve the problem of writing an .xls in Java, we used the open-source JExcelApi\textsuperscript{[13]} library. This library provided an easy to use and efficient method of data conversion once loaded on the server. Currently this reporting infrastructure is available only to Officers, and customization options are minimal, but our strategy is to provide the most information possible and let users leverage the much more powerful and elegant customization options of Excel to create the reports they want. Future improvements will most likely be restricted to finer control of what data to put on the report, and perhaps some accessibility to other roles.

The entire system is built with as loose coupling as possible using the Model-View-Controller paradigm\textsuperscript{[14]}, so that data and interface models can be extended as necessary. This will make it easier to alter and extend the interface, and extend support for the platform to new mediums, including new types of data as well as new view models, such as accessing the data through the Facebook site, or mobile phone, or doing data entry in a more aggregated fashion.

**Use Cases**

- **Rushee**: Directed to site by organization. Logs in using Facebook account and fills in missing contact info. Registers with interested houses and has access to contact info and Facebook profiles of current members to facilitate getting to know them.
- **Member**: Signs up in the same way as Rushee. Once approved by Officer as a member, has access to rushees’ information. Can leave textual reviews and numerical ratings in several categories, and browse existing reviews (which appear as anonymous)
- **Officer**: Registers house with site admin. Approves membership for other users. Can export aggregate data into Excel spreadsheet.
- **Site Admin**: Adds new schools and houses to site, registers officers for each house. Can also find the author of individual reviews. (Note: currently only implemented at SQL level)

**Conclusion**

Over the course of this project, I developed a successful web application that met the specifications laid out at the beginning of the semester. It is hosted on a stable remote server that can serve as a launching platform for future work and commercialization. A SQL database supports it that is designed to meet standards for ease of use and future extensibility. The application includes several forms for entering and altering user data, and pages for viewing individual and aggregate data of other users. The review functionality for Members over Rushees is also fully implemented. These pages are aesthetically pleasing and have been error-
checked such that all noticeable errors are resolved, exceptions are not exposed to the user, and data is verified as valid before being entered in the database. Additionally, administrative controls allow Officers to be promoted for each house who can manage routine administration through another interface. Officers can also export aggregate reports into Excel for easy distribution or further manipulation as they choose. Profiles and reviews are searchable using the standard web-search interface and an inverted-index format. Data access and modification is tightly monitored using the Role Based Access Control paradigm to ensure security and validity. Finally, the application is integrated into the Facebook Platform using the Facebook Connect technology. This allows users to use their Facebook account and import certain data from their Facebook profile, as well as leveraging an industry established authentication mechanism. It also provides an important cross-branding mechanism for commercialization (the appeal of the Facebook image) and many opportunities for future expansion and integration with other social networking media.

I was fortunate in that most of this project proceeded more or less according to plan. I had some experience with JSP and Servlets, as well as SQL, which served me well and made those elements for the most part straightforward. I had more to learn about advanced HTML (such as proper use of tables and css) and JavaScript, so those elements proved frustrating at times. The two biggest challenges were getting the site properly hosted on a remote server and properly communicating with the Facebook servers. The big commercial web hosting providers are focusing on PHP and ASP, so my initial foray with one of them led to a very frustrating week over Winter Break of trying to understand how their Java servers worked through poor documentation and customer service, and ultimately realizing that they had disabled access to a core technology that made it impossible for me to use their site. For this reason I was unable to have the site up for beta testing during spring rush at Penn, which was very unfortunate. The next hosting company I chose specialized in Java, so I was able to do what I wanted, but there were still a number of hurdles with using a Tomcat server administered by someone else and shared by other apps out of my control.

Facebook integration was also especially challenging because they had discontinued official support for Java in their platform about a year prior to this project. Most of the Facebook documentation was again geared toward PHP and JavaScript, and once I found an updated Java library, it was also poorly documented. The Connect technology is also very new (released only a few months ago) and so most of my learning for that was also by example from other sites. I had to scour the web for other resources and for several problems, just keep trying different approaches until I found a successful one. However after I figured out the basic connectivity, it was fairly straightforward to do the things I wanted to do. I did have to learn about JSON and figure out how to properly pack and unpack it for the various method calls.
While straightforward, developing the RBAC model also took a great deal of careful thought. I had to be very careful with each piece of data exposed to check that the only users viewing that data were authorized to do so. This got complicated very quickly, especially as there were two dimensions to consider, role and house/university affiliation, even after the system knew which user it was talking to. Facebook provided the authentication mechanism, but it was difficult to nail down exactly what criteria must be met in each case in order to display a data item. For this reason, debugging and checking for corner cases was also difficult.

Future Development

rushdB was developed with intent for commercialization, and there were plans all along to expand it further after this year. This appears to be on track in collaboration with Uberate Inc., a group of entrepreneurs based at UNC Chapel Hill. They have already started applying for incubator or seed funding for this project, which will accelerate now that the system is fully operational. There has also been significant interest in the Greek community there and at Penn as the system has been developed. We hope to hire an additional programmer over the summer (a student at Berkeley) to continue development in areas such as data redundancy, further Facebook integration, and improved search as well as the usual speed and reliability optimizations. We will also begin conducting usability tests in preparation for fall rush. Throughout this project I have been careful to avoid using Penn resources so that the transition to commercialization would be easy after my graduation, and this has been successful.

There are a number of challenges on the horizon as development continues. Moving into a commercialization phase will require a heavier emphasis on performance while maintaining the high guarantee of security. As development moves away from core functionality, the modularity built into the application and database will become more critical to continued success, and any weaknesses will surface and have to be dealt with. It will also take a significant investment of time to understand the context and needs of the target audience so they can be properly addressed. Usability studies will be necessary to fine tune the functionality and interfaces.
References


This is an existing Facebook application with a similar target audience as the rushdB. It has a different feature set and is not competition, but demonstrates that this is a viable sector of the population for developing a (Facebook-integrated) web application to meet their needs.


The website for the company that developed the above Application, built on their platform that is the basis for many such applications.


This is a similar web application which shares the features of anonymous polling and reviews.


Digg is a more popular site with a similar underlying structure to the rushdB and a simple interface for anonymous reviews of content, in this case articles.


This description of Role Based Access Control is provided by a government agency, NIST, and was updated in 2007. It provides a clear description of the model which was very helpful in development, and demonstrates that this is a widely accepted model that is still relevant today.


The paper which developed the RBAC ideas described in the NIST FAQ. It expands on the basic RBAC model to include a hierarchy of roles, an idea used in this project to simplify the rights scheme. It is a scholarly and peer-reviewed paper, suggesting that the findings are valid and this is a legitimate approach to authorization that has been validated by a wide range of
computer scientists. With the above FAQ, this validates that this approach is accepted in both scholarly and industry circles and should therefore be fairly robust and reliable.


This article is a description of the most recent TLS protocol. It is put forth by the Internet Engineering Task Force, a major organization supported by both industry and the government, and is recent. This validates https as a robust security protocol with a strong theoretical underpinning.


This is a description of JSP from its creator, Sun Microsystems. As the developer of Java, Sun is a trusted company, and this technology should be good to use.


This introduces the Facebook Connect technology to developers. As a major initiative created by a social networking giant, this technology is clearly supported and will continue to be developed. It is very recent; suggesting that the feature set will continue to expand. The developer audience is indicative that it should be easy to code with.


This paper introduces an algorithm for stemming to aid in text search. This is an important work that is somewhat dated but remains relevant – still taught in the Databases class at Penn. It validates this approach as legitimate and the implementation as solid since it was developed by Porter himself.


A description of XFBML by Facebook.

A description of the JSON standard.


The documentation and binaries for the JExcelApi library.


This is a description of the MVC model for application development. It is current, targeted at developers, and generated by Microsoft. This indicates that it should be an industry standard and best practice for development, and is a good model for developing rushdB.