DESIGN ANALYSIS

MITjobtalk: Timmy Galvin, Mari Miyachi, Josh Duncavage

OVERVIEW

Key design challenges

We have a number of major design challenges, which as a group we discussed and for the most part resolved in our final product. The challenges were as follows:

1. Ensuring user privacy and anonymity
2. Determining content search and guiding the user through our site
3. Generating good and trustworthy content
4. Ranking user generated content
5. Deciding on the availability of editing privileges

DETAILS

Data representation

Our object model is very similar to the representation within our database. Interviews, offers, and questions are jointly be identified by a companies and users and need either to identify them. Answers and votes are be jointly identified by questions and users, again needing either to identify them.

Key design decisions

Anonymity

The question here is how to protect a user's identity. If a user chooses to post something anonymously, will that content still be tied to their account in the back-end with that relation just being covered upfront? Or would it be better to have an "anonymous" account that anonymous content is posted under? We will probably implement the first option. While it requires us to be significantly more careful about not leaking the user's identity when displaying data, it will allow the user to go back and edit/delete that content because they still "own" it.

We also to choose to make also user's anonymous by default, with no option to reveal their identity. We operate here under the assumption that MIT email authentication would ensure a very limited amount of spam accounts, and as a result, the MITjobtalk community will have less of a need to identity information with names in order to find it trustworthy. We made the choice to prioritize the sensitivity of the information of the user choice to reveal their identity. Many users, we believe, would not even take advantage of a feature to reduce their privacy privileges, so while this may be a feature in a future iteration, we decided to exclude it.

Email confirmation

While our initial design called for certificates to verify the MIT identity of our users, we found this was impossible to implement if not deploying to Scripts. So we resorted to our backup option of MIT email confirmation. This is not nearly as secure, since MIT servlists are able to register and create active accounts. However, it is still a layer of user protection, since only users with mit.edu email adddress may even register. In order to ensure a heightened security level, email verification expires after 10 days.

Content Search
There are a lot of ways to implement a content search: regex, closest to, begins with, etc. We implemented a begins-with, case-insensitive system because our user base will be MIT students who we assume have some technical/computer knowledge and also know what company they are looking for. A begins with search is also the simplest and most resource-light approach. This proved to become a little slow once we added thousands of companies to our database, so we considered something like a fuzzy search. However, from a user perspective, when a student goes to search for a company, they know exactly what name they are looking for and thus a begins-with search will yield more appropriate results.

Pre-population and Adding Companies
Generating good and trustworthy content we knew would be a challenge from the beginning. We harvested company information from CrunchBase, to minimize the number of companies that user's would have to add and thus reducing the possibility of bad user submissions. To generate the interview and offer content, we intended to work with the Career's Office. However, it was a slow process, and we thus resorted to our backup option of running a script to populate the site. While not as valuable, since the data we populated with is more loosely based off of statistics rather than actual data points, it still is valuable to have example data off of which users can base their own submissions.

Voting System
The question here is how to treat the vote system including factors like the uniqueness of votes, how upvotes and downvotes interact, etc. The first option would be to have two vote columns for every comment in the user table to minimize the number of cross-table mappings. The second option would be having a table that just stored a vote as a user, comment, and positive or negative one value (or two columns). The first option doesn't seem to have many positives, it would require dynamicity in the table and it would be extremely space inefficient. We chose the second option because it seemed simpler in its overall construction, though we had to do a little designing to ensure a user only had either an upvote or downvote for a single element.

Admin Account
We foresaw the potential that users would create duplicate companies or companies that don't actually exist, but without the privilege to delete them, the site may be polluted. With this in mind, we implemented an admin account that had privileges to edit and delete all site content. However, this posed a huge security risk on our site, and we ultimately felt that this sacrifice was not worth it. After pre-populating our database from CrunchBase, there was significantly less risk of users entering bad company information.

EVALUATION

Design critique
From a user's perspective, our site is very successful. We were able to streamline the navigation such that the user's main goal, of finding information by company, is easily achieved from the homepage. What's more, company information is summarized at multiple points on the company landing page, such that the

user is never overwhelmed with data. If ever a user does want more information, it is easy to navigate to a page where this is displayed. Though we employed bootstrap in our design, we were able to move away from the typical look of this plugin, and our site thus has a clean, unique feel.

From a developer's perspective, we made good use of the separation between models and controllers, separating functions and commenting our code appropriately. For a site of our size and scope, we have an appropriate amount of controllers. By creating the company landing page, we were also able to streamline the views needed in our design. We made proper use of rails conventions like render, and used proper AJAX and JQuery within the site.

Our most successful design decision was implementing a quick and intuitive begins-with search. We discussed this aspect of our design thoroughly, and thought through the user perspective such that our final design represents the best option. Users go to our site with certain companies in mind, thus making fuzzy searches and the like inappropriate. Since it is very unlikely that the user will have the incorrect spelling of a company they are highly interested in interviewing with, a starts-with search, while narrow, delivers the best results in this use case.

Our least successful design decision was how we store interview questions. Instead of individual fields, interview questions are stored as a text field. Thus, we were unable to dynamically allow users to add questions to their particular interview experiences, but only list them in a large block. While this works fine in the case of one interview question, if users have more than one question they would like to share, the interface is no longer as successful. Our poor handling of interview questions is an example of a restricted object model. An object could have been made to represent interview questions, and thus allow for the dynamic addition of questions to an interview experience.

Our foremost priority for improvement is to create some way of editing content. While we did not prioritize this in our current implementation for security reasons, it would be highly valuable to have some sort of super-user class that had limited editing permissions, such as the ability to delete companies that have no content, or to flag certain user-generated content as inappropriate or spam.