MealXchange

Michael Buono (PM) mbuono@princeton.edu
Ava Chen avac@princeton.edu
Asavari Sinha asavaris@princeton.edu
Louis Guerra guerra@princeton.edu
Paco Avila favila@princeton.edu

1 Background

Each year, approximately 70% of Princeton's juniors and seniors choose to dine at one of the University's 11 private, co-ed eating clubs [1]. In order to facilitate continued interaction between friends in separate clubs, the dining system allows members from different clubs to exchange meals with each other. A meal exchange between two members of different eating clubs constitutes each person hosting the other at their respective clubs. The meal that is hosted must be the same at both clubs (i.e. breakfast, lunch, dinner, or brunch) in order to meet the exchange requirement. Each month, the number of exchanges between each pair of members must balance out, or the club which has hosted outstanding meals will charge its respective members for those meals. Club members can also host a certain number of guests per month, where these guests do not have to be members of any eating club and are not obligated to complete an exchange.

2 Problem

The current system in place for monitoring meal exchanges uses a pen-and-paper method to record and tally the meals hosted at each club. Whenever a member hosts a meal at his/her club, he/she must fill out a paper slip with the name and club of the host and guest, as well as the date and the meal exchanged. The guest must also fill out his/her social security number (something not everyone feels comfortable doing!). A meal checker who is responsible for logging all member meals at the club oversees this process. This system has proven both inconvenient and faulty, however, as members have no way of verifying the number of meal exchanges they have outstanding with friends, and club administrators spend hours manually tabulating meal exchanges at the end of each month and

[1] http://www.princeton.edu/pub/profile/campus-life/

coordinating with other clubs to tally up uncompleted meal exchanges. Maintaining a record of exchanges via paper slips is not only a hassle, but is also inevitably prone to human error in lost slips or incorrect tallies. The result is that members get unnecessarily charged for meals they do not know have not been fully exchanged by the end of the month.

3 Our Solution: MealXchange

Our solution is a Web application called MealXchange, hosted at `www.princeton-mealxchange.com`. In this system, clubs can register for accounts on our webapp. Once logged into a club account, the club can edit its membership and club preferences (such as meal times). The club's meal checker need only oversee members register meal exchanges on a dedicated computer by typing in the net ID of the host and guest, instead of filling out a paper slip (we leveraged the fact that the checker generally already has a computer out to log all member meals). The net IDs are first verified with the respective clubs' databases, and then used to send an email for the host and guest to confirm the meal exchanged. Clubs can view all of their members' exchanges on our webapp, which has a feature to download a .csv file of the computed balance of exchanges in order to simplify digital calculations, as well as a feature to send email reminders of outstanding meal exchanges on the click of a button to all members of the club.

The service also allows students to register guest meals online at the meal check station. Clubs can edit their preferences to set a certain number of guest meals allowed per month, and the system keeps track of the number of guest meals each member has hosted in the month, disallowing any more than the maximum number of guests per member that the club has specified. The functionality for meal exchanges, including viewing them, downloading the .csv file, and sending email confirmations email reminders, also extends to recording and viewing guest meals with our webapp.

4 Design Decisions and Goals

Simplicity

One of our huge priorities for this project was to keep the user interface and general concept extremely simple and minimalistic. The reason for this design decision was that we found the biggest hurdle faced by similar products in gaining traction with the eating clubs was that most clubs were reluctant to stray away from the current system, for fear of overcomplicating their infrastructure or losing desired flexibility by digitizing their records. Thus, we wanted a service that could be easily and painlessly integrated into the existing eating club infrastructure, to minimize the overhaul needed to put the system in place. For this reason, we decided to use a web application, leveraging the fact that most clubs' meal check stations already use a dedicated computer to log member meals. We also worked to create a user interface that was extremely easy to learn and use, and that would be adaptable by the current system that clubs use to collect meal exchanges. An improvement upon the current system is that our service requires only two pieces of information to perform an exchange: the net IDs of the two parties involved in the exchange. The service then does all verification and confirmation automatically. This is a far cry from all the information needed to fill out a paper slip, including both members' names and respective clubs, the date, the meal exchanged, and even the guest's SSN. A feature for clubs to change their preferences whenever desired allows for more flexibility, which basic networking with various club officers suggested was the main goal of the clubs, along with simplicity of use.

Efficiency

The biggest advantage that our system offers over the current method of recording meal exchanges is an underlying database that can be easily be accessed and viewed by eating club administrators. We decided to have one database in which each club was a single object. Each club object has associated with it three corresponding objects: a list of members, a set of preferences, and a record of all exchanges associated with members of that club. With only one database table (? fact check this), lookups would be more efficient, but from the user perspective, clubs can only view the information corresponding to their own club object. Inherently, a digitized database of meal exchange and guest meal records is more efficient to look up outstanding exchanges, unconfirmed exchanges, and number of guest meals left in the month. As mentioned, the previous pen-and-paper system offered no easy way of checking and/or verifying any of this information. (? add things about backend)

Security

Since a lot of money is at stake in potentially charging members incorrectly for outstanding meal exchanges, we decided that we needed a system that provides reliable security. The most hazardous breach of security involves recording an exchange or guest meal incorrectly, or allowing people to pose as different members of a club when registering an exchange or guest meal. Preventing this requires securely verifying a person with his/her net ID to authenticate and confirm the record. This security aspect is critical in convincing clubs that our system will not be abused for people to steal exchanges or guest meals. Our design decision for this security aspect underwent three iterations:

1) First attempt:

First, we tried to authenticate both parties in an exchange using the University's CAS login system. In a dogged attempt to successfully get this authentication to work, we spent our first few weeks of precious time fiddling with CAS login. However, we found that the CAS system is simply not designed for verifying two individuals at once (as it was created for login access and not for authentication) without unnecessarily rendering the process convoluted and user-unfriendly, with various simultaneous logouts and logins. Since we could not find an effective way to integrate a double CAS login into our code and user interface, we eventually decided to try something else.

2) Second attempt:

Instead of a double CAS login, we then tried to use a single CAS login to authenticate the host, and then try a separate means of authenticating the guest. However, this seemed to lack a streamlined user interface, with different verification methods for host and guest for no apparent reason. It also appeared more prone to abuse than a more uniform authentication process.

3) Third attempt:

We finally decided to abandon the CAS login system altogether, instead using the club membership database to verify the net IDs and email confirmation using these net IDs. Upon typing in a net ID, the system checks that the club indeed has a member with that valid net ID. Then, it sends an email to the Princeton account with that net ID in order for the corresponding individual to confirm the exchange at the click of a button. The advantage of this design decision is that it retains the security afforded by the CAS login (leveraging the individuals' unique identification with Princeton via secure University email). Yet, it allows us to forgo the clunky UI involved with dealing with CAS login and logout, as well as the difficulty integrating CAS into our system.

Transparency

One of the main goals of our proposed system is to provide transparency for both users and club administrators to keep track of meal exchanges, in order to minimize the number of unnecessary charges for outstanding exchanges at the end of the month. Thus, we wanted a system that would be able to communicate the status of members' meal exchanges and allotted guest meals whenever needed. An open system for maintaining these records would dramatically cut down miscommunications and errors that arise in the current system. For this, we also had a few different design considerations:

1) First approach:

Originally, we intended the service to be individual-based, in which our main users were the members of the clubs. In this system, members each have accounts, which they could log into to check on their running tabs. However, we realized when networking with club officials that this sort of distributed system would take control away from the clubs, and make it difficult for clubs to maintain the flexible system they currently implement. It also added a lot of unnecessary overhead to our project. We realized it was unrealistic to expect all members to adopt this method on their own time, and that with this layout, it would be difficult for clubs to keep track of a holistic picture of meal exchanges and guest meals logged by their members. Since club membership is often very dynamic, we wanted to give clubs the option to add and delete members as they chose and have it automatically reflect in their database. This would not have been possible with a distributed system where students are the main users. Since this option did not readily provide transparency to the clubs, we found that it failed in our primary goal of making our product attractive for the clubs to adopt. (? fact check this)

2) Second approach:

We decided instead to make the eating clubs our main users. This provides transparency for the clubs, as they can view all their exchanges, as well as edit their membership and preferences. In order to provide transparency to members, we decided to leverage the email confirmations we were already sending when members registered an exchange or guest meal. We tacked onto this confirmation email an update on the given member's current status regarding his/her confirmed and unconfirmed exchanges and remaining number of guest meals for the month. Clubs can also click a button to send all members this status update whenever they want, perhaps as the end of the month nears and people want to clear any outstanding exchanges. Members can also check the status of their exchanges or guest meals at the meal check station, since the club's View Exchanges page has a feature to search for members by name (? fact check this), net ID, or class year. Thus, transparency for both clubs and members is satisfied with this approach.

5 User Interface and Testing

Because simplicity and usability were such huge priorities for our team, we wanted to make sure that our user interface was as conducive to these goals as possible. We experimented with various design decisions in the UI, coupled with possible user flows to navigate our service. In the end, we went with a design that consisted of only three pages to be viewed by members, and three pages intended for club use.

The former set of pages consists of the home page, where members can select to register a meal exchange or a guest meal, and two pages corresponding to each type of meal. These three pages contain minimal content, only displaying exactly what is needed at each step of the process (aside from pretty Princeton images in the background). This is meant to speed up the user flow and make the process of registering an exchange or guest meal quick and painless.

The latter set of pages consists of a page to edit club preferences, a page to add or delete members, and a page to view exchanges. Each of these three pages features simple aesthetics and clear but concise usage options, to facilitate ease of use by clubs administrators.

For testing, we first ran through the entire process ourselves of registering a club, editing the club's preferences, adding and deleting members (individually and in bulk), and then registering meal exchanges and guest meals. We made sure exchanges were being recorded and calculated properly, emails were being sent accurately, and the .csv file was being downloaded with the correct information. After we ran through the entire user flow to test and debug, we decided to leverage the advantage that one of our members is currently an officer of Terrace, one of the eating clubs. He was able to show our product to some of the other officers and members and ask for their feedback on how the website looked and felt to the intended user. These tests on real users confirmed our hopes that the interface was simple and easy to use by both the club and its members.

6 Milestones

We were perhaps a bit too ambitious with our initial milestones. Since none of us had had a lot of experience planning out and working on a large project like this, we underestimated the time it would take to set up the backend system, create a user interface, and connect the two. We ran into some trouble in the first few weeks for two reasons.

Firstly, most of our group members experienced difficulty setting up the appropriate environment for developing and testing our code. Installing the Django framework and its necessary auxiliary packages was not as simple as we had thought. Luckily, one of our members ran a local version of Ubuntu that successfully set up the environment, so everyone worked off this one computer for our first few sessions. Although we quickly found that this wasn't the most efficient method of coding (since only one person can type at a time), working closely together to set up the foundation of our webapp proved valuable later on when we split up and developed different parts of the app independently.

Our second hurdle in the first few weeks involved the CAS system. As mentioned in section 4, our initial choice to use CAS login for our authentication process turned out to be a mistake. The way CAS is designed simply is not conducive to a double authentication procedure, which was our intended use. We spent a couple weeks trying everything to integrate CAS login into our system, and ultimately this caused us to waste a lot of time.

Because of these two issues, we were a bit slow to start actual development. However, we found that once we decided to split the work into backend and frontend, with 2-3 people working on each side, we were able to catch up to our original milestones and code more efficiently. Although this meant that not everyone was involved in the development of every part of the code, deciding to split the work ended up being a good choice that worked out well for us, because we had point people whenever we ran into trouble with any part of the code when putting it together. However, if we could do it over again, we would certainly give ourselves more time in the initial stages of development to figure out how to set up the environment properly, and also better compartmentalize the work so we didn't spend so much time trying one thing that wasn't going to work. This is definitely something I would advise future groups to be very aware of.

7 Learning Experience

As mentioned above, there were a couple of things we learned along the way as we hurdled through our milestones at various speeds.

Early in the project's life, we were very enthusiastic about the choice to use the CAS login system to verify our exchanges. We knew that in the history of this class, many groups had used CAS login successfully for their apps, so we figured that any trouble we ran into could be solved by consulting with TAs, professors, or students that have had experience with CAS login in the past. However, we overlooked a key difference between the goals of our implementation and the intended usage of the CAS system, which was that CAS supports single use login, whereas we needed a system for double use authentication (effectively, login/logout). This use was simply not supported by the current CAS design, and our group made the mistake of spending too much time attempting to mediate this conflict instead of brainstorming other ways to approach the problem. Without realizing it, we had developed tunnel vision in stubbornly trying to integrate CAS into our system when we should have been looking at other solutions entirely. We were surprised when, after discussing with various TAs and professors in different departments, none of these people knew how to implement CAS the way we had hoped. This was the impetus for us to finally choose another course of action.

Our decision to use email authentication actually turned out to be one of the ones that worked out well for us, and we are very glad we went with this choice. Not only did it allow us to forgo the clunky UI of including CAS login while still affording the same University-established authentication, but it also gave us the opportunity to add whatever info we wanted in our confirmation emails. We decided to include an update on the status of meal exchanges and guest meals for a member whenever we sent that member a confirmation email. This improved our goal of transparency for the club members.

Another good choice we made around was to outsource our user login and registration system (that clubs would need to create and log into their accounts) to an open-source Django package that handled authentication for us. The package is called Registration-Redux, and it was written by James Bennett and Andrew Cutler [2]. It was quite a pleasant surprise for us to find that such a package existed open-source for us to use!

Registration-Redux offered a secure system that allowed people to easily create and access their accounts, and it integrated very well with our Django framework. However, the way that the package handled the creation of the secure forms (automatically with a random token, hidden away from the client) meant that our group had limited flexibility in making design choices for many of our login and registration forms. To this end, we did our best to modify our user interface to integrate into the given style of these forms, while still fulfilling our goal of creating an uncluttered, minimalistic user interface.

Our experience in working with Registration-Redux was a good reminder that open-source, third party applications can be very useful to work with. If we had attempted to write our own code for registration and login, we would have spent a lot of unnecessary time coding something that had already been done a lot more efficiently. This was also a good learning experience in figuring out how to integrate open-source code into our framework, and to work with not only our own written code, but other people's code as well, which is something that happens very frequently when coding ”in the real world.” We would definitely advise future groups to not be afraid to look into open-source options for some parts of their development, which might be better handled by existing packages. That way, they can spend more time developing the nuances of their application instead of working on certain features that have already been coded up multiple times before.

(? add things about backend)

8 Possible Improvements and Future Work

We propose four future avenues of improvement in our project:

Firstly, it would be useful to have some sort of club re-authentication in order to access the set of pages intended for club use (i.e. Club Preferences, Edit Membership, View Exchanges). Currently, we do not have this feature implemented because we wanted a very streamlined user interface, without the hassle of re-logging in every time something needs to be changed. We decided instead to only display the hamburger button giving access to these pages on the club's home page, and not on the pages members view to register a meal exchange or a guest meal. Although this should ensure that members do not even see an option to click the menu button, it still is not the most secure method of ensuring that people cannot mess with the club's settings.

[2] `https://django-registration-redux.readthedocs.org/en/latest/`

In addition, if we had more time to improve our product, one of the first options we would tackle would be the creation of a mobile app that would integrate into our system of logging exchanges. We think that the convenience gained from allowing users to sign up for exchanges via their phones would be a huge boost to the appeal of this product. To this end, we would have to figure out a way to integrate the current club-oriented usage system with a member-oriented usage system. This would involve a fair amount of authentication protocols, and we would have to be very aware of possible security breeches in opening up the use of our product to students instead of offering it only to clubs.

Thirdly, although we believe it was a good idea to use club membership verification and email confirmation to authenticate our exchanges and guest meals, there is still room for improvement in this authentication procedure. Perhaps instead of University CAS login, we could integrate Registration-Redux in a way to give members their own accounts to register and view exchanges. As mentioned above, shifting our focus to add a member-based dimension to our system would add another layer of complexity that would have been too difficult to address in our limited time: we would have to have a reliable and up-to-date cross-club database that links student accounts to different clubs. Since club membership is so dynamic, it would be hard to keep thousands of member accounts up-to-date, creating and destroying accounts as members join and drop different clubs (? fact check this). But this is certainly and area of improvement that can be looked into.

Finally, a big improvement that we are aware of lies in integrating our MealXchange system with the system the University dining halls use to track exchanges with the eating clubs. Currently the eating clubs operate almost entirely independently from the dining halls, and they log meals in a completely different way. Meal exchanges between members of eating clubs and students on a dining hall plan are recorded by University dining hall protocol. It was not possible for us in our 10-12 weeks to successfully move through the various layers of bureaucratic protocol in changing the way the University dining halls track meal exchanges. However, in the future, the eating clubs and the University dining halls may want to consider sharing a common database for logging their meals, and perhaps establish a more integrated system for students with dining hall plans to trade meals with eating club members. This would open the door to opportunities for a more connected campus. We believe that, if University protocol ever decided to integrate its meal exchange system with that of the eating clubs, MealXchange has great potential to connect these two spheres of student life.

9 Acknowledgments

We would like to thank Kelvin Zou for his continued input, advice, and guidance throughout the course of this project's development. We also want to extend our gratitude to Professor Brian Kernighan for establishing a friendly, open environment in this course and for giving us the opportunity to work on such a neat hands-on, open-ended project. This experience provided incredibly valuable insight into the nuances of real-world software development, which is something we really appreciate. Thank you!