MealXchange: Internals

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

1 The Database

We are using Django's default SQLite database with the following models, each of which is represented in a separate table.

1.1 Exchange (host name, host club, guest name, guest club, breakfast, lunch, dinner, brunch, month)

For each pair of students that meal exchange, we store two *Exchanges* objects. These two objects store the same information, but in a slightly different way. For example, if Ava, a member of Terrace, and Michael, a member of Tower, perform a meal exchange, an *Exchanges* object will be created in which Ava is the host and Michael is the guest, and an *Exchanges* object will be created in which Michael is the host and Ava is the guest. We do this so that when an eating club wants to view all its members' exchanges, we can return this information easily. For example, when Terrace wants to view its members' exchanges we will return *Exchanges* object in which Ava is the host, and Michael is the guest.

The *Exchanges* objects keep track of outstanding meal exchanges with counter variables for each meal. For example, after Ava hosts Michael at lunch at Terrace, the *Exchanges* object in which Ava is the host will have its ”lunch” counter *increase* by 1, while the *Exchanges* object in which Michael is the host will have its ”lunch” counter *decrease* by 1. The +1 in the *Exchanges* object in which Ava is the host represents that Ava, the host, has now hosted Michael, the guest, and is owed a lunch at Michael's club for the meal exchange to be even. Similarly, the -1 in the *Exchanges* object in which Michael is the host indicates that Michael owes Ava one lunch at Tower in order for the exchange tab to be even.

Exchanges objects only last 12 months; at the beginning of each month, all *Exchanges* objects that are 12 months old are deleted from the database. We do this so as to avoid cluttering the database and interface. Furthermore, because each club only really is interested in exchanges that happened in the past month or maybe two months, when we output an Excel file for the clubs to download, we include one sheet that has the exchanges for the current month, and one sheet that contains the exchanges for the past month.

1.2 ConfirmExchange (hostHasConfirmed, guestHasConfirmed, host, host club, guest, meal, month, hostConfirmString, guestConfirmString)

When two students sign up for a meal exchange, we create a *ConfirmExchange* object. This object is essentially a placeholder for the exchange until it is confirmed by both students via email. The object contains two 64 bit alphanumeric strings: one for the host confirmation link, and one for the guest confirmation link. For the exchange to be logged in the database, both the host and the guest must visit the confirmation URL (which is unique for each meal exchange and for each person): `http://princeton-mealxchange.com/Xchange/Confirmation/<confirmation_string>`. Once the host visits their confirmation URL, the *ConfirmExchange* object's ”hostHasConfirmed” field will be set to true. Similarly, once the guest visits their confirmation URL, the *ConfirmExchange* object's ”guestHasConfirmed” field will be set to true. Once both the ”guestHasConfirmed” and ”hostHasConfirmed” fields are set to true, the meal exchange is logged in the database (in other words, an Exchanges object is either created or modified), and that *ConfirmExchange* object is deleted.

In short, the *ConfirmExchange* object allows us to keep track of and confirm individual meal exchanges, even though the Exchanges object only logs the ”tab” of two students who have meal exchanged.

1.3 ConfirmGuest (hostHasConfirmed, host, hostConfirmString)

The *ConfirmGuest* object functions exactly like the *ConfirmExchange* object, except there is only one confirmation field: ”hostHasConfirmed”. Once the host visits their confirmation URL, the guest meal is confirmed, the host's number of guests is decremented, and the *ConfirmGuest* object is deleted.

1.4 Members (name, club, year, netID, numguests)

Whenever an eating club adds a member, we create a corresponding *Members* object with all the relevant details pertaining to a member of a club, including how many guests that member is allowed each month. A *Members* object is deleted when clubs delete members through our web interface, and finding the *Members* object associated with a student is easy because each university student has a unique netID. When a *Members* object is deleted, its corresponding exchanges are not deleted by design, because any outstanding exchanges involving that member should still be kept track of.

1.5 ClubPrefs(name, breakfast hours, lunch hours, dinner hours, brunch hours, max_guests, last_login)

We prompt each eating club to enter the start and end times of each meal offered at that club, so that we do not need to rely on members to reliably enter which meal they are exchanging. We also receive as input the maximum number of guest meals per month at that club, so that we can reset the guest meals of all members of that club at the beginning of each month. The *last_login* allows us to keep track of when it is a new month, and thus when it is time to both update guest meals and delete year-old *Exchanges* objects.

2 Development Environment

Some of our users may be Windows users and others may be OS X users. We therefore used an environment that was portable and not dependent on a certain operating system. We used the following languages: Python, JQuery, HTML, and CSS. We used a Django framework, SQLite database and hosted the web application on Amazon Web Server (AWS).

3 Running MealXchange

MealXchange can be run locally provided the following modules and applications are installed — Python (2.7.6), Django (1.8), django-registration-redux (1.1), django-tables2 (0.16.0) and xlwt (1.0.0). These can easily be installed using the pip package which should be included if you are running Python 2.7.6 or later. Our main server is AWS and requires a few additional modules to run — apache2, libapache2-mod-wsgi, mysql-server and python-mysqldb, which can all be installed using the command

```
sudo apt-get install <package_name>
```

We push to our AWS from a git repository and SQLite creates a folder which stores our databases.

4 System Operation

Django has a file called urls.py (located in the ”polls” directory) that tells the framework which method in views.py corresponds to each url. In views.py, each method handles a ’view’ or a page of our application. We do basic computations within each of these methods and create the necessary forms for the pages they correspond to. The view then calls render() which loads the relevant html template for that page with the necessary forms. The html templates import the relevant css files to style the page and format our forms. Each form has a POST method which, when the submit button is clicked, returns to the same url and therefore the same view. However, because it’s a post method, we can retrieve the data stored in the form and do the relevant processing. Once this is complete we can redirect to another url or simply reload our current view and repeat this process.