eXchange: a COS 333 Product Guide

James Almeida Emanuel Castaneda Meaghan O'Neill
Sumer Parikh Danielle Pintz

9 May 2016

Introduction

This guide is divided into two sections: a user guide and a developer guide. The user guide gives a very detailed overview of how to use the various components of our iOS app and our webapp, while the developer guide details the way we went about developing both apps.

The iOS student app was modeled from a very strong prototype which can be found at: https://jamesalmeida.proto.io/share/?id=21c47824-e34e-460b-82ea-9b173da86e97v=6. Therefore, the iOS is very client focused, with an outcome of it being reasonably intuitive to use. The User Guide will cover all of the typical screens so that a user can explore its full functionality. The club app, which is hosted as a website, was made taking into account a lot of input of meal checkers at certain eating clubs, and is therefore also simple and intuitive to use. We will explore the functionality of the webapp in this guide. The User Guide is divided into two sections: one focusing on the iOS app, and another focusing on the webapp.

The Developer Guide will also be divided into two sections for the iOS app and the webapp.

User Guide

iOS Application for Students

Anyone with a Princeton netID who has access to an iPhone or iPad can login to our app using their netID and password (see Figure 1 in the Appendix). Once logged in, the user will see the navigation bar at the bottom of the app, with four buttons: "eXchange", "News Feed", "My Meals", and "User". The iOS app is extremely easy to use and navigate using these 4 basic navigation buttons.

Request an eXchange (see Figure 2 in the Appendix)

Clicking on "eXchange" will let you request, accept, reschedule, or decline a meal with any Princeton student in an eating club other than your own. Once on the "eXchange" page, the user will see two tabs at the top: "Request Meal" and "Pending Meals". "Request Meal" is the page that shows up by default when you click on "eXchange", and it shows your best friends (those people with whom you eXchange the most) and underneath that it shows all of the Princeton students in eating clubs. There is a search bar above best friends which allows you to search through the entire database of users, and updates the list of matching names in real time. To request a meal with someone, simply tap on the student you wish to meal exchange with. A meal exchange schedule view will be displayed that allows the user to select a date, an eating club, and a meal type. This request is then sent to the students "Pending Meals".

Pending Meals (see Figure 3 in the Appendix)

When a user gets a request from someone else, it shows up in his/her "Pending Meals" tab, where he can take one of three actions: accept, reschedule, or decline. If the user accepts, the meal will appear under both students' "My Meals" tab, under "Upcoming". If the user elects to reschedule, a meal exchange schedule view will be presented, allowing the user to create a new request (with a new date and meal type) and send it back to the other student. This action will remove the previous request to the user from the database and send the new request to the other student's "Pending Meals". If the user declines the meal, it will be deleted from his pending meals.

Newsfeed (see Figures 4 & 5 in the Appendix)

The second main button on the bottom navigation bar, "Newsfeed", contains a meal exchange newsfeed, which brings a social component to the app. The newsfeed displays the meal exchange activity of all Princeton eating club members. By selecting the "My Club" tab at the top, the user can view meal exchanges that have occurred at their eating club. By selecting the "Princeton" tab at the top, the user can view all meal exchanges that have occurred in the entire

eating club network. Users also have the ability to "like" a meal by tapping on its "like" button.

My Meals (see Figures 6, 7 & 8 in the Appendix)

In the third main button on the bottom nav bar, "My Meals", the user can see his/her incomplete meal exchanges by selecting the "Incomplete" tab up top, his/her upcoming scheduled meal exchanges by selecting the "Upcoming" tab up top, and his/her past exchanges by selecting the "Complete" tab. "Incomplete" also tells the user how many days left he has to complete the exchange, and if he clicks on the meal exchange it will bring up a new page for him to send a meal request to the other student.

User (see Figure 9 in the Appendix)

In the fourth button on the bottom nav bar, "User", the user can view his profile. Basic information about the logged in user, including their name, netID, club, and a user image are displayed. If the user has not set a profile picture, a default image of the Princeton Tiger is shown. Users can set their profile picture by tapping on the "Change Picture" button. This button presents an image picker of the user's photo library. Selecting an image presents a scale and crop view that allows users the ability to edit the selected image. A copy of the new image is set as the user's profile picture by tapping the "Done" button. Finally, we have a "Log Out" button that logs the user out of the application and redirects to our CAS login view.

Webapp for Meal Checkers

Our webapp is hosted on github, and can be accessed at the following URL: sumerp.github.io/clubapp (see Figure 10 in the Appendix). Each of the 11 eating clubs has its own email/password login, as listed below:

Cannon: cannonmealexchange@gmail.com/dialelm
Cap Gown: capmealexchange@gmail.com/andgown
Charter: chartermealexchange@gmail.com/engineers
Cloister: cloistermealexchange@gmail.com/floatersandboaters
Colonial: colonialmealexchange@gmail.com/asiansandfriends
Cottage: cottagemealexchange@gmail.com/friendsandriches
Ivy: ivymealexchange@gmail.com/est1879
Quad: quadmealexchange@gmail.com/lawnparties
Terrace: terracemealexchange@gmail.com/foodislove
Tower: towermealexchange@gmail.com/theater
Tiger Inn: timealexchange@gmail.com/theglorious

Once a club is logged in, no matter which page it is currently on, it will see 6 buttons in the navigation bar at the top of the page: "Home", "Meal Check", "Create a Meal", "Members List", "View Exchanges", and "Logout".

Home (see Figure 11 in the Appendix)

Clicking on "Home" will bring you to the home page, which simply welcomes you to the web app, tells you which club you're logged in as, and gives you the contact info of all 5 members of our group so that you can contact us if you have questions. This is the page, as one might guess, that you see when you first log in.

Meal Check (see Figure 12 in the Appendix)

Clicking on "Meal Check" gives you a platform with which to confirm meal exchanges that have been pre-scheduled on the iOS app. You simply choose the type (breakfast, lunch, or dinner) and date of the meal for which you want to check exchanges, and click the orange button that says "View scheduled meals". All of the pre-scheduled meals made on the app (listed with the host's netID, guest's netID, meal date, and meal type) will then show up, with an option to "Confirm" or "Cancel". Click "Confirm" to verify that the member listed as the host did, in fact, bring the listed guest to the meal, and click "Cancel" if this pair does not show up.

Create a Meal (see Figure 13 in the Appendix)

Click on "Create a Meal" to create exchanges if two people show up at a club having forgotten to pre-schedule a meal exchange, or if a person does not have an iOS system on which to pre-schedule exchanges. (Alternatively, if a person does

not have an iOS system, he/she can log into eXchange on someone's iPhone/iPad to pre-schedule.) "Create a Meal" is a page with a form in which you enter the host's netID, the guest's netID, the type of meal, and the date of the meal. Click "Submit" to log the meal.

Members List (see Figures 14, 15 & 16 in the Appendix)

Clicking on "Members List" will give you a dropdown menu with the following options: "View Member List", "Add a Member", and "Delete a Member". Clicking on "View Member List" will give a list of all of the members (including their netIDs, full names, and PUIDs) in the club that you're logged in as. Clicking on "Add a Member" will bring you to a form that asks for a new member's name, netID, and PUID. Clicking on "Delete a Member" will bring you to a page that lets you delete a member from the member list simply by entering the person's netID. (These features will normally will be used to either add new sophomore members after they bicker/sign-in or delete senior members when they graduate, but sometimes people drop clubs before graduation or join clubs as juniors or seniors.)

View Exchanges (see Figures 17 & 18)

Clicking on "View Exchanges" will give you a dropdown menu with the following options: "Complete Exchanges" and "Incomplete Exchanges". Clicking on "Complete Exchanges" will show you a list of all of the exchanges that have been completed by the members of the club that you're logged in as. Clicking on "Incomplete Exchanges" will show you a list of all of the instances that a member has brought a guest to meal but hasn't eaten at the other club. This will be used to charge members at the end of each month if they haven't completed an exchange.

Logout

At any time, a club can log out by clicking the "Logout" button in the top right corner.

Developer Guide

Preliminary information on our Firebase

Firebase is a no-sequel database that stores information in JSON dictionaries. Our Firebase consisted of the following sub dictionaries within the master dictionary:

- complete-exchange : Used to store a list of the completed meal exchanges that each student has completed. One separate entry for both members who complete a meal exchange. Must be created or confirmed by a meal-checker.
- friends : Used to store a list of netids that are associated with a "friend score." The friend score is an indicator of how many meal exchanges one user has requested with another user. This is used to place emojis next to people's names in the eXchange tab.
- incomplete-exchange : Used to store a list of the incompleted meal exchanges that each student has partaken in. One separate entry for both members who have started but not completed a meal exchange. Must be created or confirmed by a meal checker.
- newsfeed : Contains sub dictionaries of all the meals that have been either created or confirmed to have happened by a meal checker. Used for the newsfeed in the iOS app.
- pending : Contains sub dictionaries mapping a student's netid to all the meals that other students have initiated but haven't been confirmed, rescheduled, denied or cancelled by the student.
- students : Contains sub dictionaries mapping a student's netID to her name, netID, club and PUID.
- upcoming : When meals have been confirmed in the iOS app, details of the meal are added to upcoming sub-dictionary mapped to the netID of both students who will be eating together.

The URL where one can visualize our database is https://princeton-exchange.firebaseio.com/ (NOTE: This cannot be viewed unless you are added as a collaborator. If you wish to be given permission to view and edit the database, email emanuelc@princeton.edu)

iOS App Developer Guide

Development Environment

We developed our app in Xcode, using a storyboard to create the layout of the app, and linked it to Swift files that implement functionality.

Dependencies

We use CocoaPods to handle our project dependencies. CocoaPods is a dependency manager, built in Ruby, for iOS projects written in Swift or Objective-C. Our project has two dependencies: Firebase and RSKImageCropper. Our general use of Firebase was previously discussed and more details of its implementation in our project will be given in further sections. RSKImageCropper is an image cropper for iOS, which we use to support a user's ability to change their profile picture. Both projects are added to our project by simply including them into our project's podfile.

Student CAS Login

We use CAS authentication for user verification. Our app presents a webview directed towards a CAS protected website. Once the user successfully passes verification the webview is redirected to a landing page where the user's netID is gathered. The webview is then dismissed and the user's netID is passed to our main tab bar controller.

Data Structures

We have three main data structures in our implementation. The first is a "Student" data structure which stores a student's name, netID, club, prox number, friend score (for determining best friends), and image. The second is a "Meal" data structure which stores the host, the guest, the date, the type of meal (lunch or dinner), and the number of likes (for the newsfeed). The third is an "eXchange" data structure which stores both Students and both Meals involved in the exchange, and whether the exchange is completed or not. Each data structure is stored in the database as a key-value dictionary where the instance variable names are the key and their values are stored as strings.

Database Interaction

- **Create a Request** When a user creates a new request with another user, this request is added to the database under the second user's pending.
- **Accept a Request** When a user accepts a request with another user, this meal is added to both user's upcoming, and deleted from the user's pending in the database.
- **Reschedule a Request** When a user reschedules a request with another user, this request is added to the database under the second user's pending and deleted from the first user's pending in the database.
- **Decline request** When a user declines a request with another user, this request is simply deleted from the user's pending in the database.

- **Newsfeed** The newsfeed retrieves data from the database from the newsfeed section and puts it into one large array of Meals. It then displays every element in the array in a tableview. For the Club filtered data, we create a new array by iterating through the large array and only using Meals from the user's club. We display every element in this array in a tableview, when the "My Club" button is selected.

- **My Meals (incomplete, upcoming, complete)** My meals retrieves data from the database from the incomplete-exchange section, complete-exchange section, and upcoming section and displays it under the My Meals tab view. When the user clicks on a cell in the incomplete exchanges section a new page opens allowing him to create a new request with the person to complete the exchange. If the user makes the request, it will be added to the other person's pending section in the database.

- **User Profile Pictures** Each new user is initialized with an empty value for its image in our database. Our app then renders the user image as the default Princeton Tiger. If a user updates their profile picture the image is saved as a Base 64 encoded string. This data can then be decoded by our app and converted into a proper image to display.

Webapp Developer Guide

While writing the code for the webapp, we consciously made an effort to ensure that it would be easy to maintain. Since Firebase has such a robust javascript library, we decided against using a backend framework to access the database. Maintaining the webapp only requires knowledge of Firebase (which we have provided above), as well as familiarity with its javascript API which can be found at https://www.firebase.com/docs/web/api/

Platform Overview

Local Development

All of the local development was done exclusively with HTML and javascript files in one directory (since most of the CSS was handled with bootstrap, the remaining CSS elements were directly coded into the HTML pages). Each view of the webapp is an HTML page with a corresponding javascript page. The HTML page describes what the page does (for example 'Create a Meal' is names 'create-meal.html). The corresponding javascript page shares the name of the HTML page with Sc appended at the end (for example 'createMealSc.js'). There is one exception to this–the HTML page for the login page is called index.html, but its javascript page is called login.js. While developing, pages were individually run and checked in chrome, using the chrome JavaScript Developer Console (found under views → Developer → JavaScript Console). All developments were done locally and pushed to github pages (for the iOS team to work with) every time significant functionality was added.

Production

Since we did not use a backend framework, and the entire website is built on HTML, CSS and javascript, we hosted the webapp using github pages. If the website is picked up, we will buy a custom domain name and host it there using github.

Dependencies

The dependencies of the webapp are the following:

- Twitter Bootstrap for CSS styling: accessed online at `https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css` and `https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap-theme.min.css`
- Twitter Bootstrap javascript plugin: accessed online at `https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js`
- jQuery javascript library: accessed online at `https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js`

- Firebase javascript plugin: accessed online at `https://cdn.firebase.com/js/client/2.2.1/firebase.js`
- Any web-browser capable of running HTML5 (read: 'any web browser a typical user would use)

Frontend and templating

The front end template of the webapp was done exclusively using HTML and CSS. Bootstrap components were used, but no external theme was used, and we designed the webapp layout throwing bootstrap elements together in a way that we thought looked good. Since most of the other styling was only to center bootstrap and HTML elements, we did not write separate CSS stylesheets but just coded the CSS elements within HTML style tags in the head of the HTML document.

For each page, navbar is the same (apart from recognition and indentation of the page that it is on), and the rest of the general template is the same.

Firebase and Javascript as a Backend Framework to add functionality

When creating a new feature of the site, we followed the following protocol:

1. We would first create a HTML page for it to be displayed, and make sure that the HTML page is accessible through the navbar, and that this page is given the class element 'class="active"', so that bootstrap would be able to recognize that this is an active page that could be navigated to. We would also have to be careful to ensure that this was added to the HTML navbar elements of other HTML pages, so that this page could be navigated to from other pages.

2. We would write a general template for the page, into which information could be drawn for the database. There were two general types of pages that we had to write. Pages from which information would be logged into the database, and pages from which information would be drawn from the database. For information-logging pages, we would create a form. For information-drawing pages, we would create a table header, and a div element to which we could append the information that we retrieved from the database that is relevant to that page.

3. Once we were satisfied with the general layout of the page, we would start interacting with the database using Firebase's javascript API in order to add or extract data from the database. The following are some salient features of this interaction that are common to almost all the javascript pages (barring perhaps home.js):

   (a) Authenticate the user (using Firebase) to make sure that only registered users can use the system and so that we know which club is logged in to retrieve the relevant data.

(b) We would establish a reference of the specific database dictionary that we wish to reference.

(c) Either log the data to the database for a information-logging page, or store a snapshot of the relevant data into a variable and iterate through it in order to extract relevant information. It is important to note that we switched from firebases forEach method to iterate through the data to the standard javascript 'for (object in object)', so that we could keep access the dictionary keys that we were iterating through.

(d) If it was a information-drawing page, we would append it to the appropriate ¡div¿ tag in order to append the appropriate information to the relevant table.

The only page that slightly differed from that template was the Meal check page. This would follow the above template in order to retrieve the relevant data, but would then also have to log information about the meal to the database depending on whether it is confirmed or denied. In effect, it would perform two cycles of the above protocol when used to its full potential.

N.B. because of the asynchronous nature of Javascript and Firebase, we had to code many things in large loops instead of modulating the different aspects into separate functions. We tried, but modularity simply did not work because they are asynchronous.

Appendix

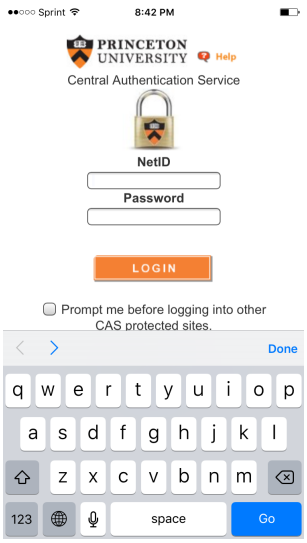

Figure 1: Login Screen

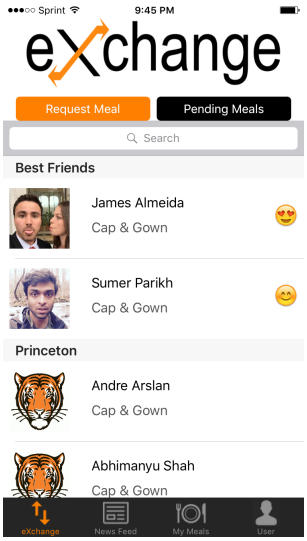

Figure 2: eXchange Request

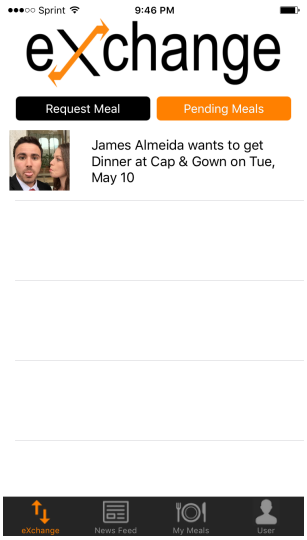

Figure 3: eXchange Pending

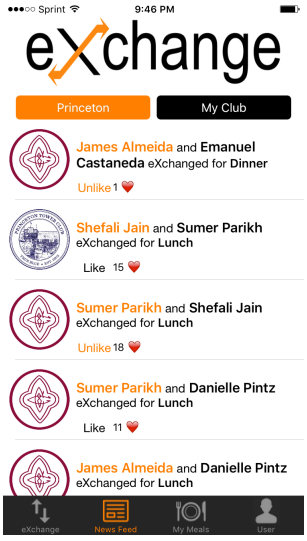

Figure 4: Princeton Newsfeed

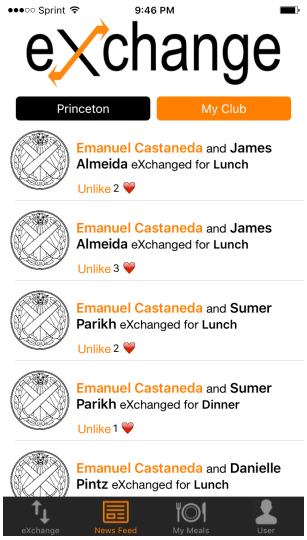

Figure 5: Club Newsfeed

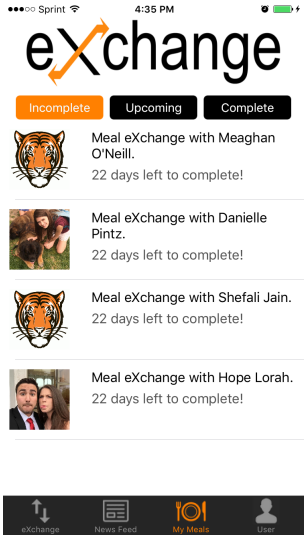

Figure 6: Incomplete Meal Exchanges

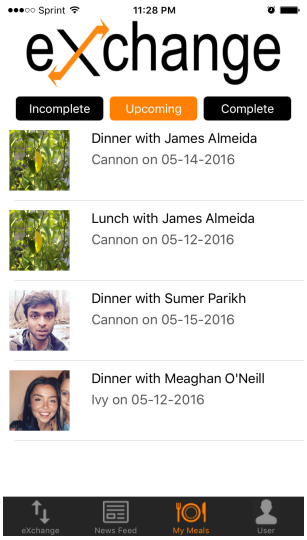

Figure 7: Upcoming Meals

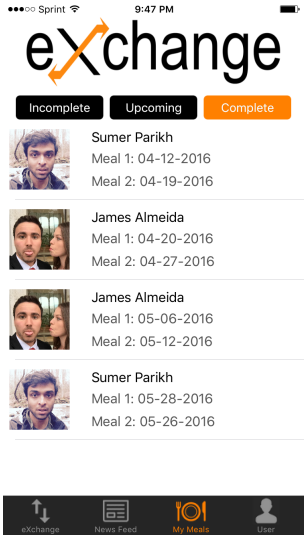

Figure 8: Completed Meal Exchanges

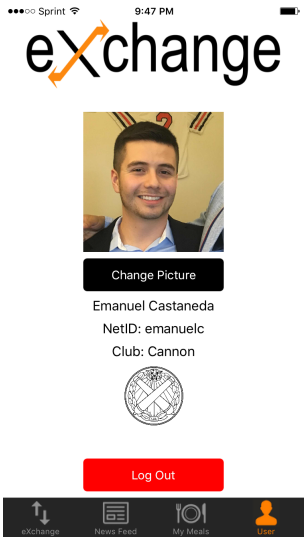

Figure 9: User Page

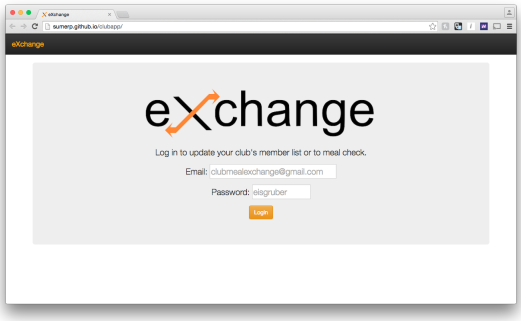

Figure 10: Login

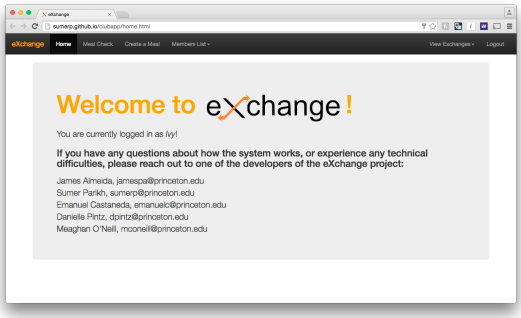

Figure 11: Home

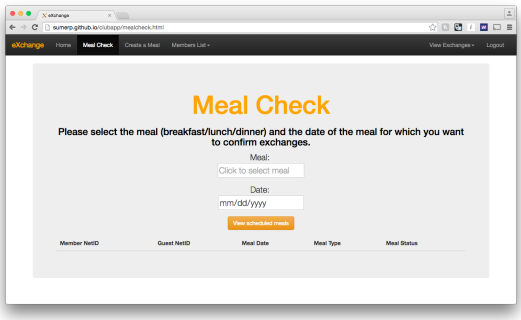

Figure 12: Meal Check

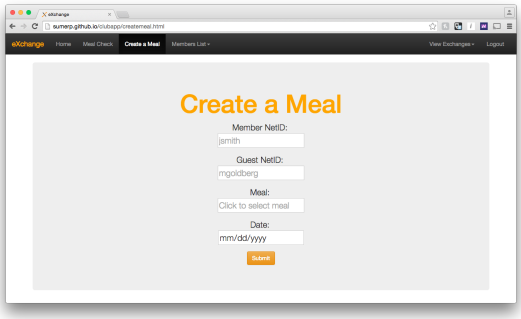

Figure 13: Create a Meal

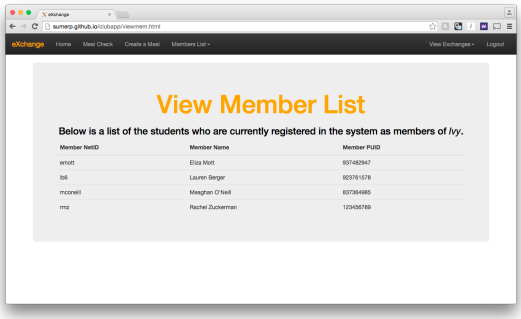

Figure 14: View Member List

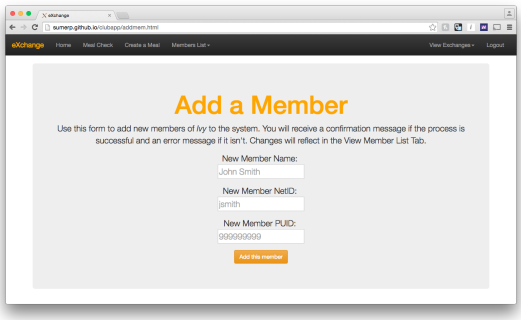

Figure 15: Add a Member

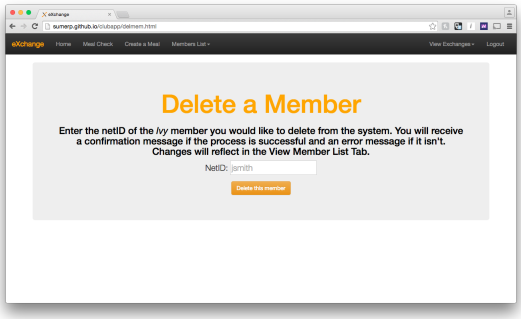

Figure 16: Delete a Member

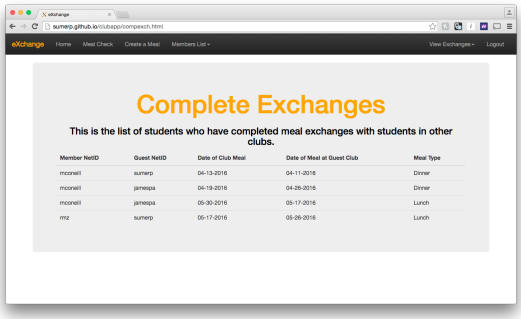

Figure 17: Complete Exchanges

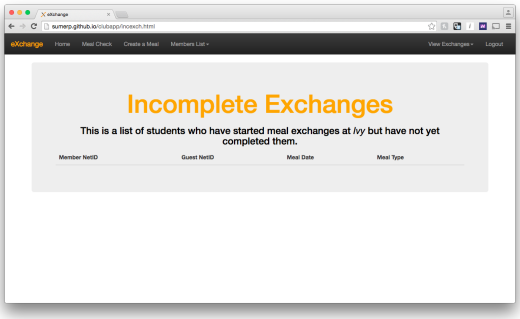

Figure 18: Incomplete Exchanges