GroSCery

Complete Documentation

By: Sarah Cheung, Naman Kedia, Pinghao Luo, Maxwell Newman, Kristof Osswald, and Natalie Riopelle

Table of Contents

Project Proposal

Members

- Natalie Riopelle: nriopell@usc.edu
- Naman Kedia: nkedia@usc.edu
- Maxwell Newman: mdnewman@usc.edu
- Kristof Osswald: osswald@usc.edu
- Sarah Cheung: cheungsa@usc.edu
- Pinghao Luo: pinghaol@usc.edu

Weekly Meetings

Team Meetings:	6 – 8 P.M. on Wednesday at Leavey Library Basement
C.P. Meetings:	10:50 – 11:20 A.M. on Thursday at Grace Ford Salvatori Hall

Project Proposal

Our project will be an iOS grocery tracker app that will simplify how the occupants of an apartment pay for the various shared accoutrements of the apartment. The occupants will be able to specify which apartment items they use and therefore should have to pay for (for instance, all members of the apartment use soap and paper towels, but only some members may drink milk or eat apples). Then, when one member of the apartment buys a particular item at the store, they can log the price of the item in the app, and the app will disseminate the cost of the item among those who use it. Ideally, the app will be able to periodically notify the participants of the balance they owe other members of the suite, or possibly will just be able to send the necessary Venmo requests.

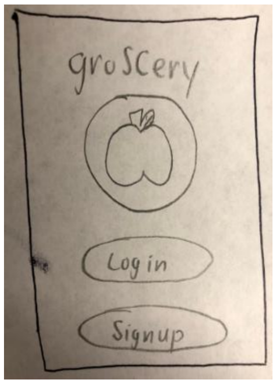
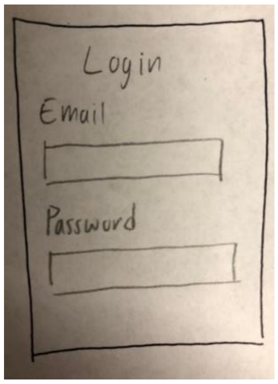
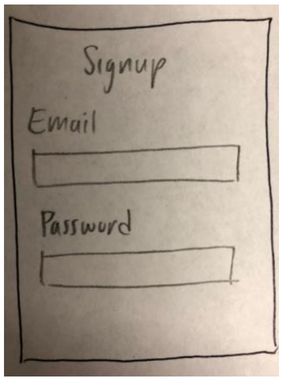
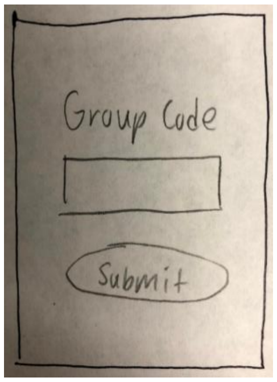
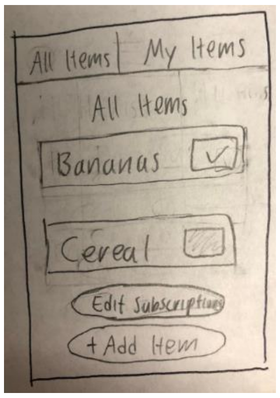
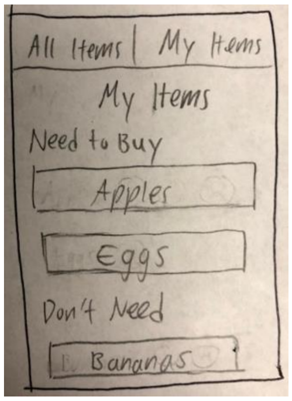
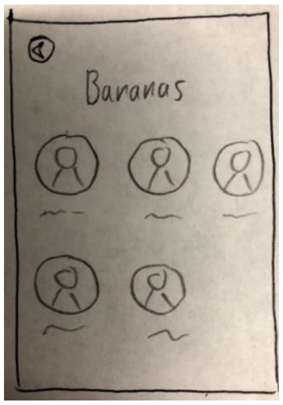
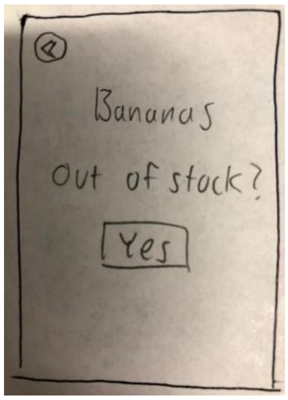
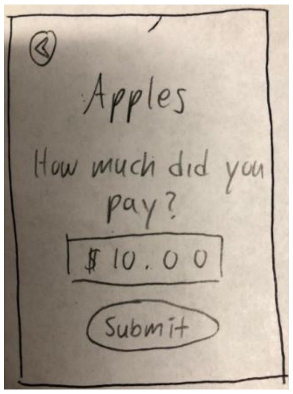
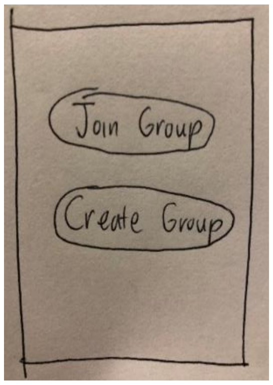
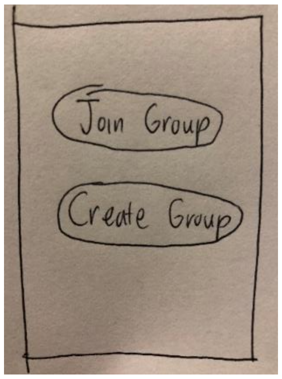
The user will interact with the app entirely through a GUI designed for smartphones. The app will need a way to allow a user to select the items they've purchased and enter the price of those items (probably using a list of checkboxes and a textbox, respectively). The GUI will therefore consist of many inputs which are compatible with touchscreens. This app will also rely extensively upon reliable networking, since changes to each users' balance on the app will be dictated by the items and prices entered by users across each phone, and balances need to be consistent between the apps of every user. Moreover, the app will use multithreading to load information such as which items the user subscribes to on the background thread while updating the UI of the app on the main thread.

High-Level Requirements

For our application, we are going to be building an iOS app that will interact with Java Web Server as an API. The app should allow users to log in and join a group. Each group will have a list of shared items for their apartment. Individual users can subscribe to certain items that they use in their apartment. Those who are subscribed to an item will be notified when it “goes out of stock”. The next person who buys that item will then be able to input a price and the app will automatically divide that price up among those subscribed. At the end of the week, each member will be sent a notification for how much they owe each member of the group.

The app will allow users to make changes to which items they are personally subscribed to along with adding items to their grocery list, if necessary. The following outlines what a user’s experience might look like:

- **Home Page**: The first time a user opens the app, they will have to login or create an account. Once signed in, they can choose to “Create a new group” or “Join a group.” They will either be given a new unique group code or asked to provide a code in order to join an already existing group, respectively. Note: a group would compromise of an apartment that shares groceries, rent, etc.
    - **Subpages**: Login, Signup, Group Subscription, Join Group, Create Group
- **Group Page:** After you join a group, the user will be taken to the Group Page, which will have a list of current products that other members are subscribed to. Each of these products will have a list of users subscribed to that item, displayed in a Subscription page. A user can choose to subscribe to any item in order to share the cost. They will also be able to add new items to the group list. The user can edit this list at any time in order to add or subscribe to new items.
- **User Page:** The page will have a list of items that the specific user is subscribed to, separated by what is out of stock and what is in stock. When something goes out of stock (which a user can input through this page), a notification will go out to each user that the item is out. When a user buys an item, you can check it off, record the price for which you bought it, and the app will disburse the cost of the items to the people who use it. Each time a needed item is paid for, the app will send a notification to each user subscribed to that item (excluding the user who paid for it) detailing how much money they owe and to whom.
    - **Subpages**: Item Subscription, Item Stock, Payment

Figure 1: Home 	**Figure 2**: Login 	**Figure 3**: Signup 	**Figure 4**: Join Group 
Figure 5: Group  All Items My Items All Items Bananas Cereal Edit Subscriptions + Add Item	**Figure 6**: User  All Items My Items My Items Need to Buy Apples Eggs Don't Need Bananas	**Figure 7**: Item Subscription 	**Figure 8**: Item Stock 
Figure 9: Payment 	**Figure 10**: Group Subscription 	**Figure 11**: Create Group 	

Technical Specifications

iOS App On-Boarding Process (8 hours)

- Home Page
    - This is the user landing page when they first open the app.
    - This page gives users the option to login to an existing account or sign up to create a new account.
- Signup Page
    - If a user signs up, their information will be stored by the backend mySQL infrastructure, which can be used later tok verify subsequent logins.
    - After signing up, the user will be prompted to search for and join an apartment group (or create a group, if they're the founder, which would send them to the Create new group page).
- Login Page
    - This is the page where a user who is already subscribed to the app will sign in.
    - It will be connected to our backend mySQL infrastructure, which will verify the user's login information.
    - If login is successful, the user will be pushed to their group page. If login is unsuccessful, an error message will be presented.
- Create Group Page
    - This is the page where users will be able to create a new group if they want to make one for their apartment.
    - It will be connected to our backend mySQL infrastructure, which will add the new group code.
    - It will generate a new unique group code for the user, automatically associate the user with this new group, and insert the code into the database.
- Join Group Page
    - This is the page where users can join an existing group.
    - It will be connected to our backend mySQL infrastructure, which will verify the group code.
    - The user enters a group code in the text field, and if valid, the user will be redirected to the group page. Otherwise, an error message will pop up.

<u>iOS App Main Functionality (10 hours)</u>

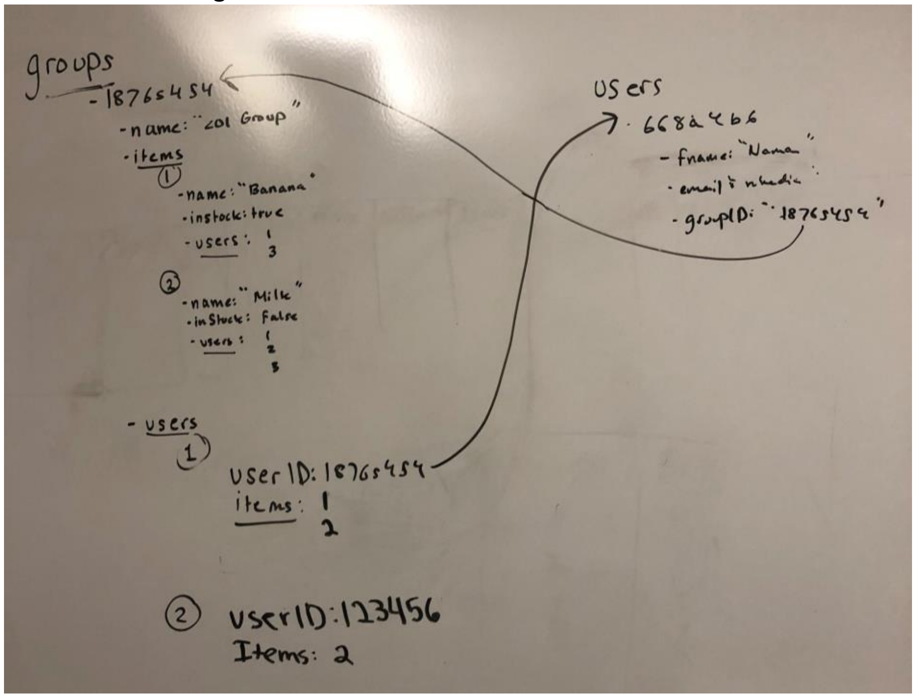
- Group Page
    - This is the main page for a user. After a user has signed up/logged in, they will be able to examine the items to which they are subscribed within their group.
    - This will be accomplished by query backend servers for a list of items associated with the current user. The list of items will display information about the item, such as the last time it was purchased, and if it is out of stock.
    - There will also be a button for adding a new item to the list of items used by the apartment, or for subscribing to another existing item within the apartment.
    - This will be a table view with two sections, one for items that are “In Stock” and one with items that our “Out of Stock”.
    - The user will be able to mark an item out of stock from the “In Stock” list and indicate that they bought an item from the “Out of Stock” list, which will send a money request to all users in the group who are subscribed to that item.
    - Add new item function
        - This function will be used for anyone in the group to add an item that the group does not yet subscribe to.
        - The user will have a text field to insert the item name and an image if they like.
        - The app will add the item to the users list of subscribed items and also the list of total items in the groups list in the database.
    - Subscribe item function
        - This function will allow a user to subscribe to any items listed on the group page.
        - There will be a list of items and they will be able to tap the ones they want to subscribe to.
        - These items will then be added to the user’s list of items.
- User Page
    - This page will have a list of items that the specific user is subscribed to, separated by what is out of stock and what is in stock.

    - When a user buys an item, you can check it off, record the price for which you bought it, and the app will disburse the cost of the items to the people who use it.
    - Each time a needed item is paid for, the app will send a notification to each user subscribed to that item (excluding the user who paid for it) detailing how much money they owe and to whom.
    - Subpages: Item Subscription, Item Stock, Payment

<u>Database (8 hours)</u>

- We are going to have 2 main tables implemented in our database using mySQL
- We will have a table for Groups, which will have every group listed by their unique group ID.
    - When you open up the group tab, it will have their name, itemID, and usersList.
        - Items will have a name, an “in Stock” boolean, and a list of userIDs
            - ✓ The userID is only for those who are subscribed to that item.
        - Users will have List of Items
            - ✓ This list holds only the items that each user in the group is subscribed to.
- The other table will be the User table
    - It will have Users, listed by IDs.
        - Inside the Users table will be their information including name, email, phone number, and groupID.
            - ✓ This groupID is how we will identify which group a user is in once they login to their account.

- Here is an image of what the tables will hold:



Benchmark Requirements (8 hours)

- *2 weeks from now*, we want to have our full database set up and connected to the app.
  - We will have the tables fully worked out so that we can start creating a user interface.
- *3 weeks from now*, we want to have the sign-up page and login page working and connected to our database.
- *4 weeks from now*, we want users to be able to form groups and add items.

Detailed Design

Hardware and Software Requirements

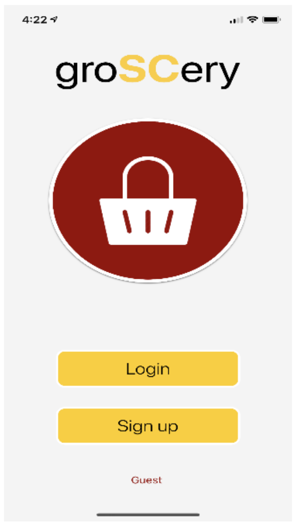
- Languages
    - Java: used to send a confirmation email to the user that he/she signed up on our app
    - PHP: used for most of the backend, including listing the groups items and whether or not the current user is subscribed to each item, allowing a user to subscribe to new items, allowing a user to see only the items he or she is subscribed to and allowing the user to click on an item either mark it as out of stock or submit how much he/she bought the item for
        - If the user submits that he/she bought an item, they can put in the price they bought it for and each user that is subscribed to that item will get a push notification to Venmo the user that bought the item.
    - Swift: used for Front End Development of each page - a photo of the layout of each page is shown below

- Database
    - MySQL

- Hardware
    - iPhone
    - MacBooks

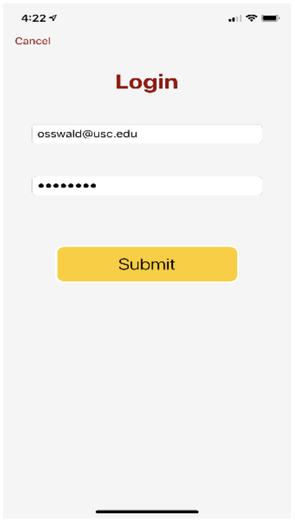
Page Layouts

Home Page (1)



- Login and Signup buttons redirect the user to the respective Login (2) or Signup page (3)

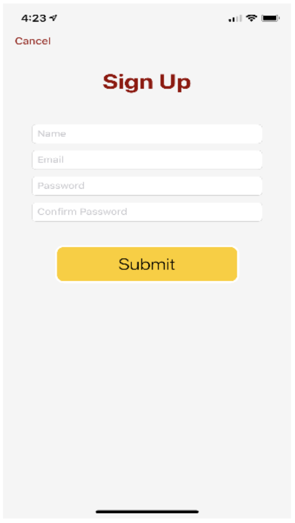
Login Page (2)



- The UserAuthentication class will access the Users table in a MySQL database named groSCery using MySQL functions called in Swift.
  - The authenticate() method will take two strings as parameters, representing the user's email address and password
  - It will return a Boolean value specifying whether the user was authenticated or not against the Users table

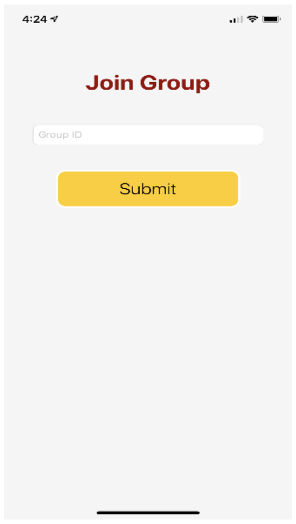
---

Signup Page (3)



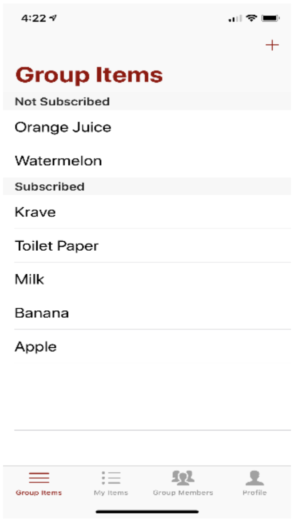
- The Signup class will access the Users table in a MySQL database
    - The checkUnique() method will take in a string as a parameter, representing the user’s email
    - It will return a Boolean value specifying whether the email was unique or not
    - If it returns true, a new User will be added to the Users table with the input email and password
    - After a successful signup, the user should be taken to the Group Subscription page (10).

Join Group Page (4)



- The GroupAuthentication class will access the Groups table in a MySQL database named groSCery using MySQL functions called in Swift
    - The authenticateGroup() method will take one string as a parameter, representing the input group code
    - It will return a Boolean value specifying whether the group code is valid or not against the Groups table; if it is, it will add the user to the specified group in the Groups table
    - After clicking the Join Group button (10), the user will be redirected to this page. There will be a form that takes in a group code string.

Group Page (5)


4:22

Group Items

Not Subscribed

Orange Juice

Watermelon

Subscribed

Krave

Toilet Paper

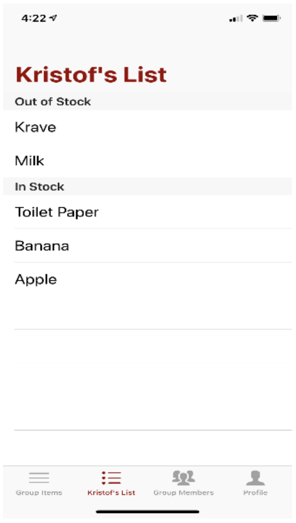
Milk

Banana

Apple

- After successfully logging in (1), joining a group (4), or creating a group (11), the user will be redirected here.
- At the top of the page will be a header that allows to the user to see all items in the group (current page) or only the items he/she is subscribed to (6)
- When the page loads, it will iterate through the Items class in the database. It will check each item to ensure there are users currently subscribed to that item. If so, it will display that item.
    - For each item, it will iterate through the users to check if the current user is subscribed to that item. If so, it will display a checkmark next to it.
- The user will be able to edit their subscribed items by clicking the “Edit” button at the bottom of the page. This will call a function that will allow the user to then check/uncheck items and then it will update the database to reflect the changes. (Will need to update the users and items table)

User Page (6)


4:22

Kristof's List

Out of Stock

Krave

Milk

In Stock

Toilet Paper

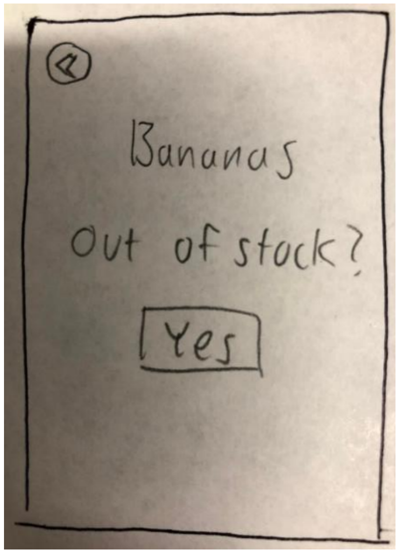
Banana

Apple

- Clicking “My Items” in the header bar redirects the user to this page
- We will access the Users table in a MySQL database named groSCery using MySQL functions called in Swift
    - The getItems() method will return a list of items that the user is subscribed to
    - The checkItems() method will iterate through the list, checking the boolean inStock in the Items table, and place each item in their respective spot under either the “Need to Buy” section or the “Don’t Need” section
- Clicking items under the “Need to Buy” section will redirect the user to the Payment Page (8)
- Clicking items under the “Don’t Need” section will have the Item Stock Popup window (7) appear

---

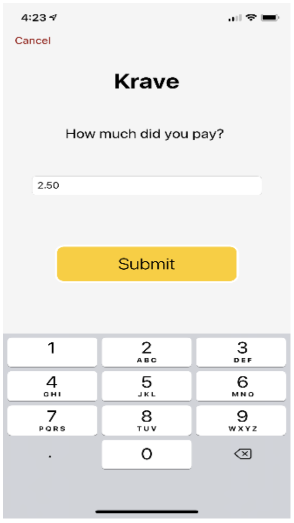
Item Stock Popup (7)



- This popup window will appear on the User Page (6)
- It will ask the user if the item is out of stock
    - The “Yes” button will access the Items table and change to boolean inStock to False

---

Payment Page (8)

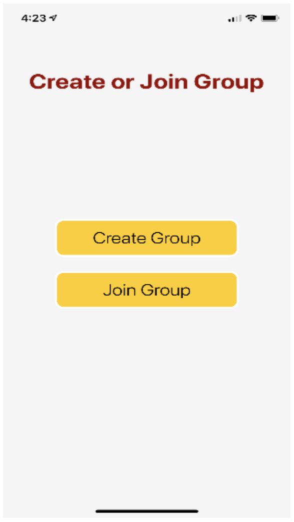


- The arrow button in the top-left corner will redirect the user to the User Page (6)
- This page will ask the user how much he/she paid for this item
    - The user can input in the text field of a form the payment amount

- Clicking the Submit button will automatically send a notification to all users subscribed to that item (except the user who paid) how much each person should pay back the user
    - The calculatePayment() method will divide the total payment amount by the number of users and return a Double (how much each person should pay back the user)
- After a successful submission of the payment amount, a confirmation message "Sent payment notifications successfully" will appear on the user's page

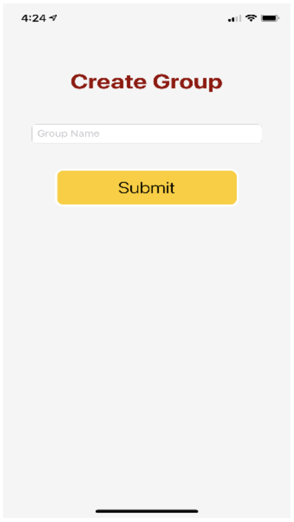
---

Group Subscription Page (9)



- The "Join Group" button should redirect the user to the Join Group Page (4)
- The "Create Group" button should redirect the user to the Create Group Page (10)

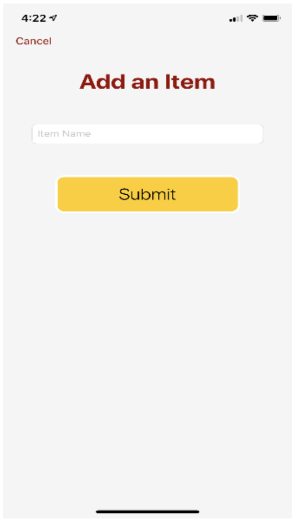
Create Group Page (10)



- The arrow button should redirect the user to the Group Page (5)
- The createGroup() function will create a new unique group code, add it to the database, and return the code as a string

---

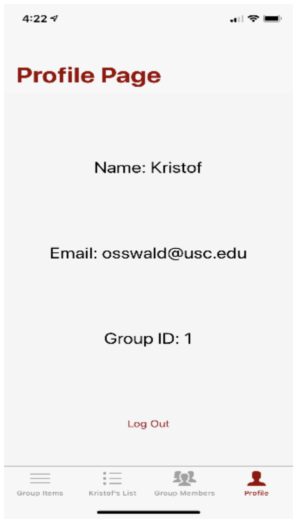
Add Item Page (11)



- The “Add Item” button from the Group Page (5) will redirect the user to this page, where he/she can add a new item for the group if it does not already exist in the database.

- This will add an item to the group's list of items and add the item to the users subscribed list. It will also automatically subscribe the user himself/herself to that added item
- **Note**: When the page loads, a function is called to ensure each item has users subscribed to it. Only items which have currently subscribed users will be shown. (Therefore, if everyone in the group unsubscribes to an item, it will no longer be shown on the page)

Profile Page (12)


4:22

Profile Page

Name: Kristof

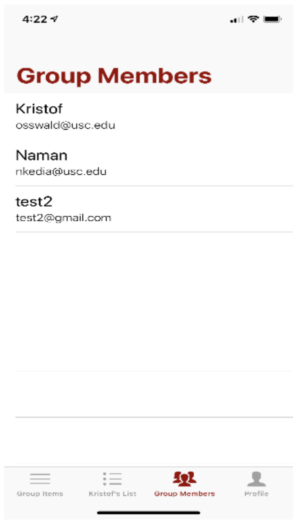
Email: osswald@usc.edu

Group ID: 1

Log Out

- This page will displays a specific user's information.
- The function GetUserInfo() will access the Users table to pull information from a specific user and return that user's information.

Group Members Page (13)



Group Members

Kristof
osswald@usc.edu

Naman
nkedia@usc.edu

test2
test2@gmail.com

- This page will display the users who are in the group.
- The function GetGroupMembers() will access the Group table and return a list of users registered in the group.

<u>Class Diagram/ Database Schema</u>

Groups Table

Group_ID	Group_Code

Users Table

User_ID	Password (Hashed)	Group_ID (Foreign)

Items Table

Item_ID	Group_ID (Foreign)	In_Stock (boolean)	Photo

Subscriptions

Subscription_ID	Item_ID (Foreign)	User_ID (Foreign)

Class/Inheritance Hierarchy

3 Classes: These are basically stored in our database rather than actual classes so that we can easily update the Groups information in real time

- Group
    - Groups have users and items
- Users
    - Users have a list of items and a Group_ID
- Items
    - Items have a list of users that are subscribed to them and also a Group_ID

Method and Variable Descriptions

- Boolean authenticate(String email, String password): checks whether the user was authenticated or not against the Users table
- Boolean checkUnique(String email): checks whether the signup email was unique or not; if it is, adds a new user to the Users table with the input email and password
- Boolean authenticateGroup(String code): checks whether the input group code is valid or not against the Groups table; if it is, adds the user to the specified group in the Groups table
- String createGroup: creates a new unique group code, adds it to the database, and returns the code as a string
- void onLoad(): iterates through the Items class in the database, checking each item to ensure there are users currently subscribed to that item
    - If so, it will display that item and iterate through the users, checking if the current user is subscribed to that item
        - If so, it will display a checkmark next to that item; otherwise, an empty checkmark box is shown
- void edit(): gives the user the functionality to check/uncheck the item boxes
- void submitEdits(): updates the groSCery database (the Users and Items tables) based on the submitted edits and reloads the Group Page to display these changes

- void addClick(): displays a popup box, containing a form, on the Group Page; includes a text field for the user to input the item he/she wants to add, and a submit button
- void addItem(String item): upon submission of the form in addClick(), this function is called and adds an item to the list of items in the Groups table
    - Also adds the item to the list of items the user is subscribed to in the Users table
- void List<String> getItems(): returns a list of items that the user is subscribed to
- void checkItems(List<String> items): iterates through the input list of items, checking each item's boolean InStock value in the Items table; based on that value, each item is placed in either the "Need to Buy" section or the "Don't Need" section
- List<String> void getUsers(): returns a list of users that are subscribed to that item
- void inStock(): accessed the Items table and changes the boolean inStock for the specific item to False
- Double calculatePayment(Double amount): divides the input amount by the number of users subscribed to that item and returns a Double for how much each person should pay back the user who purchased the item
- void submitPayment(Double amount): calls calculatePayment() and sends a notification to all users (except the user who paid) subscribed to that item to pay back the user who paid with that returned Double

Algorithms

- When we add users and groups, these will be short insert statements into our Users and Groups tables respectively.
- For the Group Page, we will have to do two select statements to get each item in from the Group based on whether or not the user in subscribed to that item.

```
SELECT FROM Items i, Users u, Subscriptions s

WHERE u.userID = currUser //currUser is a local variable

AND u.GroupID = i.GroupID

AND s.userID = currUser


SELECT FROM Items i, Users u, Subscriptions s

WHERE u.userID = currUser //currUser is a local variable
```

```
AND u.GroupID = u.currGroup
AND s.userID != currUser
```

- For the User Page, we will also have to do two select statements in order to sort by whether or not an item is out of stock.

```
SELECT FROM Items i, Users u, Subscriptions s
WHERE u.userID = currUser //currUser is a local variable
AND u.GroupID = i.GroupID
AND s.userID = currUser
AND i.inStock = FALSE

SELECT FROM Items i, Users u, Subscriptions s
WHERE u.userID = currUser //currUser is a local variable
AND u.GroupID = i.GroupID
AND s.userID = currUser
AND i.inStock = TRUE
```

- We are connecting our mySQL to our Swift code through a server with the name 201.Kristofs.app

Testing

<u>Test Case 1</u>

White Box Test – Test the login functionality by specifying an email address that exists and a password that does not match. The user should be taken back to the Login page with a message “Invalid login.”

<u>Test Case 2</u>

White Box Test – Test the login functionality by specifying an email address that does not exist. The user should be taken back to the Login page with a message “Invalid login.”

<u>Test Case 3</u>

White Box Test – Test the login functionality by specifying an email address that exists and a password that matches. The user should be taken to the Group page (if the user already joined a group).

<u>Test Case 4</u>

White Box Test – Test the login functionality by specifying an email address that exists and a password that matches. The user should be taken to the Join Group page (if the user did not already join a group).

<u>Test Case 5</u>

White Box Test – Test the signup functionality by specifying an email address that exists. The user should be taken back to the signup page with a message “User already exists.”

<u>Test Case 6</u>

White Box Test – Test the signup functionality by specifying an email address that does not exist. The user should be taken to the Join Group page.

Test Case 7

White Box Test – Test the joining group functionality by specifying a group code that does not exist. The user should be taken back to the Join Group page with a message “Invalid group code.”

Test Case 8

White Box Test – Test the joining group functionality by specifying a group code that does exist. The user should be taken to the Group page.

Test Case 9

White Box Test – Test the creating group functionality by clicking the “Create Group” button. The user should be taken back to the Create Group page with a new group code.

Test Case 10

White Box Test – Test the creating group functionality by clicking the “Create Group” button. The database should have a new unique group code added to its Group table.

Test Case 11

Black Box Test – Check if all the group’s items are correctly displayed correctly on the Group page.

Test Case 12

White Box Test – Test the edit subscription functionality by clicking the “Edit Subscription” button and clicking an unchecked box of a listed item. A check should appear in the box.

Test Case 13

White Box Test – Test the submit edits to subscription functionality by clicking the “Submit Edits” button. The user should remain on the Group page with checks next to the newly subscribed items and blank boxes nest to unsubscribed items.

Test Case 14

White Box Test – Test the add item functionality by clicking the “Add Item” button and specifying an item that already exists in the popup window. The user should remain on the Group page with the popup window and the message “Item already exists” in the window.

Test Case 15

White Box Test – Test the add item functionality by clicking the “Add Item” button and specifying an item that does not already exist in the popup window. The user should remain on the Group page with the popup window gone and the newly added item under the “All Items” list of the Group page.

Test Case 16

White Box Test – Test the remove item functionality by having all users unsubscribe to a listed item on the Group page. The item should be removed from the list on the Group page.

Test Case 17

White Box Test – Test the remove item functionality by having all users unsubscribe to a listed item on the Group page. The item should be removed from the list on the Group page.

Test Case 18

Black Box Test – Test the subscribed users functionality by clicking an item on the Group page and being redirected to the Item Subscription page. All of the users subscribed to that item should be listed.

Test Case 19

White Box Test – Test the subscribed users functionality by clicking an item on the Group page and being redirected to the Item Subscription page. All of the users subscribed to that item should be listed.

Test Case 20

White Box Test – Test the payment functionality by submitting the payment for an item from the “Need to Buy” section on the User page. All of the users (excluding the user who submitted the payment) should receive an email notification for how much they owe the user that paid.

Test Case 21

White Box Test – Test the item stock functionality by clicking “Yes” on the Stock page for an item. The user should be redirected to the User page with the item move from the “Don’t Need” section to the “Need to Buy” section.

Test Case 22

Unit Test – Insert SQL code in the username variable of the UserAuthentication.authenticate() method. Verify that the SQL code specified is not executed against the database.

Test Case 23

Unit Test – Write a Get method in frontend to received the token for authentication users. Different users should have different tokens.

Test Case 24

Unit Test – All the functionality in the frontend need a token for permission. When the frontend sends a request without a valid token, the backend should send back a “Denied” message. When the frontend sends a request with a valid token, the backend should send the proper information back.

Test Case 25

Unit Test – Write SQL to add, update and delete element in the Users, Groups, and Items tables. Verify that all the databases are functioning properly.

Test Case 26

Unit Test – Use Postman to test all the APIs for the backend:

Group:

Get:

API to list all the groups and their IDs

A group's subscribed items

Post:

Subscribe a user to a group

Create a new group and subscribe to it

Items:

Post:

Create a new item and subscribe to it

Subscribe a user to an Item

Unsubscribe a user to an Item

Purchase an item and notify group

Login, Logout, and Registration:

Post:

Login a user

Logout a user

Register a new user

User:

Get:

Get a user’s basic information test

Get a user's subscribed items

Deployment

Server

In order to deploy groSCery's backend, a publicly available server is needed for the iOS app to communicate with. There are many different ways to deploy the backend depending on your choice of OS, but for this example, we will assume you're using Ubuntu 18.04 LTS on a DigitalOcean virtual private server.

1. Deploy an Ubuntu 18.04 LTS instance
2. Connect to the machine via SSH
3. Install Nginx
4. Install MySQL
5. Install PHP
6. Install a SSL Certificate via Certbot
7. Install Composer
8. Clone repo into /var/www directory
9. Set up environment variables to connect with MySQL
10. Install all dependencies
11. Set up keys for Authentication
12. Migrate database
13. Configure Nginx
    a. Set up virtual server on port 443
    b. Provide SSL Certificate paths
    c. Point root to public folder of the project
    d. Grant permissions to project folder

Once it has been verified that the server is set up correctly, the app can be submitted to the app store.

App Store

To deploy groSCery, we would submit it on the App Store so anyone with an iOS device could search for it and download it.

1. Export the app icon in all different sizes required for the App Store and add them to Xcode.
2. Prepare screenshots. Each app can have up to five screenshots and three video previews. Prepare these screenshots and previews for all different screen sizes.
3. Test the application to make sure it is ready to be launched.
4. Visit App Store Connect and login with Apple ID
5. Create a new application on App Store Connect
6. Upload all screenshots and previews, select which category the app belongs to (utilities) in our case, select pricing, and write a description of the app.
7. Upload the app binary from XCode to App Store Connect by clicking on Archive from Xcode’s menu and clicking “upload to App Store”.
8. After the build is validated, submit the app to the App Store through App Store Connect and wait for Apple to approve the app.
9. Once it is approved, we would ensure that the app can be downloaded and properly run on an iPhone.
10. Once verified, anyone can go to the App Store, search for “groSCery”, and download the app.
11. The app should also be downloaded by the creators and tested again to ensure that the app store release is fully functional.

The app is now released.