

First Iteration First Report: mycheapfriend.com

COMS 4156: Advanced Software Engineering

Team: CheapSkates

Team Members

Double Trouble: Waseem Ilahi (wki2001@columbia.edu)

Shaoqing Niu (sn2385@columbia.edu)

Dynamic Duo: Michael Glass (mgg2102@columbia.edu)

Huning "David" Dai (hd2210@columbia.edu)



The CheapSkates from MyCheapFriend.com will lend you our **First Progress Report**, if you give us back feedback next week.

2. Revised Requirements: (Updated Priorities)

Functional Requirements

1. Accounts (High)
 - . Administrative as well as regular users.
2. Messaging (High)
 - . E-mail and text message based UI.

Non-Functional Requirements

1. Security (High)
 - . Very simple. Address Parsed from the text or email.
2. Compatibility (medium)
 - . Messages must be readable with multiple options (txt/html).
3. Usability (Medium)
 - . Easy to read.

Use Cases

The Almanac is a system for settling small debts amongst friends. The following use cases will describe the complete functionality of the system as well as rank different features in terms of their importance.

Account Management

1.1) Creating an account (high)

In order to bill friends, you need to create an account. Creating an account involves texting new_account@mycheapfriend.com. new_account@mycheapfriend.com responds with a unique e-mail address to text to and use as a security token. The text looks like this, "your unique address is <gop1bi@mycheapfriend.com> Please add it to your address book (as CheapFriend?)". This authenticates users against their unique e-mail address and treats their phone number as a unique user-id.

1.2) Changing a Password (medium)

If your password has been compromised or you want to change your security token for any other reason, you can text reset_pass@mycheapfriend.com from the cell phone that is associated with the account. The old security token will be marked, "disabled" internally and any e-mails to its address will respond with an error telling the user to update their security token or e-mail reset_pass@mycheapfriend.com.

1.3) Unsubscribing from MyCheapFriend.com (medium)

If you are either a user who wants to disable your account, or a bill-ee who does not want to receive any notifications from mycheapfriend.com, any text to unsubscribe@mycheapfriend.com will add the reply-to address to a block list and will not contact them again.

1.4) Re-subscribing from MyCheapFriend.com (medium)

If you would like to re-enable access to MyCheapFriend.com after unsubscribing, a text to resubscribe@mycheapfriend.com will remove you from the block list.

Billing a friend

2.1) Requesting a bill

2.1.1) Identifiers

2.1.1.1) The cell phone identifier (high)

In a message, when referring to another user, you may use a straight cell phone number. The following regex will validate a cell phone number. In our first version, we will only validate 10-digit cell phones numbers. `\d{10}|(\D?\d{3}\D\d{7})|(\D?\d{3}\D\d{3}\D\d{4})`

2.1.1.2) Nickname identifier / adding and changing (medium)

Each user will have his/her own table of nicknames associating self-assigned nickname strings with their friends' cell phone numbers. Attaching a nickname to a friend's cell phone number involves e-mailing your unique e-mail address a message with a cell phone number (validated with the regex above) and a nickname (validated with the following regex). Any subsequent texts with a single cell phone number and a single nickname will result in a reassignment of that nickname. `[a-zA-Z]{2,2}[a-zA-Z0-9_-]{1,8}` validates a nickname.

Note nicknames are at least 3 chars long. Nicknames are stored in a case-insensitive manner, so a cell phone number associated with "Jacob" will be able to be referred to as "jAcOb" or any other permutation.

Whenever a text is received with a nickname not in the nickname table, the text will be rejected and an error message will be returned, giving the malformed / unassociated nicknames.

2.1.2) Requesting a Bill

2.1.2.1) Requesting a bill with a single friend (high)

Billing a friend with a cell phone number involves sending a text to your unique e-mail address with an identifier and an amount (a regex validating an amount is listed below) separated by whitespace. The amount or identifier can come first. `"\${1,4}(\.\d{2})?"`, is a regex that validates an amount. The largest bill we will accept is \$9999.99.

`\s*(((identifier)\s+(amount))|((amount)\s+(identifier)))\s*` is a regex that validates a bill with a single friend. The length of the entire message must fit in a text message, i.e., 160 chars.

2.1.2.2) Billing multiple friends simultaneously (medium)

Billing multiple friends simultaneously involves sending a whitespace delimited list of amount-identifier pairs (also separated by whitespace) to your unique e-mail address. The following regex validates a message billing multiple friends `\s*(bill-with-single-friend)(\s+(bill-with-single-friend))+\s*`

Again, the message cannot exceed 160 chars

2.1.2.3) Splitting a bill with multiple friends (medium)

If you want to split a bill evenly with multiple friends? Send whitespace-delimited list of identifiers and a single amount to your unique e-mail address. To include you yourself amongst the split, also add the string "me" to the list. This will split the bill one-more way, i.e., if you are billing 4 friends \$50 and include "me" in the message, it will bill each friend \$10.

`\s*(((identifier|me){2,}(amount))|(((identifier|me){1,}(amount)(identifier|me){1,}))|((amount)(identifier|me){2,})))\s*`

Validates a bill split amongst friends (and/or yourself)

2.2) Receiving a bill (high)

When a friend bills you, you receive a text that tells you "Your cheap friend #{friend's nickname followed by cell phone # or just cell phone if you haven't assigned friend a nickname} just said that you owe him/her 20 bucks. If this is true, reply "Y" to this text. Otherwise, ignore this message."

A note about all of the following situations: MyCheapFriend.com will stagger all billing requests from different users by 2 hours. It will cancel and compound billing requests from a single user. For instance: If I bill Jacob twice within 24 hours, and he doesn't reply to the first message, the second message will include the sum of the two bills. If I bill Jacob once and Rina bills Jacob immediately after mine, Rina's message to Jacob will not be delivered to him for 2 hours after mine is delivered.???????I dont think we should compound it, as the user may accept only one of the two bills and reject the other.(he think only one of them is correct.) But not compounding two bills may lead to confusion about which bill the user responds to.

2.2.1) For registered users (medium)

2.2.1.1) Accepting a bill (medium)

Nothing special for registered users, a simple "Y" reply will do the magic.

2.2.1.2) Rejecting a bill (medium)

If no "Y" is sent back, after waiting for 24 hours (or for the next message from mycheapfriend.com to come through, that is, a minimum of 2 hours), our system will send a "Request refused: #nickname/phone number #amount" message back to your friend, who claimed that you owed him/her money, he may harass you again by replying "Y" if he insists.

2.2.2) For non-registered users (low)

2.2.2.1) Accepting a bill (low)

A non-registered user can also accept a bill by replying "Y", however, our system will send an additional message asking for the user to register. Something like: "Try out the greatest service ever - mycheapfriend.com; reply "R" to activate your account". If the user decided to accept the invitation, the process will be identical to 1. Creating an account.

2.2.2.2) Rejecting a bill (low)

The mechanism is just like for registered users.

Settling a bill

3.1) Settling a bill by the bill-er (medium)

After the user's friend pays him/her back, he sends a message with an identifier and a negative amount to clear the bill. Note: A user might also use this to offer money to his/her friends.

3.2) Settling a bill by the bill-ee (medium)

If a user bills a friend who owes him/her money, and that bill is accepted, any balance between the two users is settled before a new "debt" is accrued. i.e., if I owe Rina \$5 and I bill her \$3, then I still owe her \$2. If I then bill her \$5, she then owes me \$3

Bill report

4.1) Getting a bill report (low)

A user can check the status of all pending bills by texting "REPORT" to his unique email address.

4.2) Modify/Remove record from report (low)

A user may remove incorrect bill record from the database using its index in the report.

Web Interface (Low)

If we have the time, an alternate web-interface will be created. Users will not be able to create an account from the web interface, but will be able to log-in with their cell-phone# and unique identifier. From the web interface, they will be able to create and settle bills as well as check on all pending and completed bills.

1. Work Breakdown (Revised)

TA's Comments: "Work Breakdown: 2.5/5

You need to explain clearly in the work breakdown how each section of your application fits into the component model framework and why. Also, please try to format the work breakdown

in some way that directly aligns with the requirements (so each requirement is clearly covered by one or more work breakdown items). "

Not finished(Shaoqing was working on this) (we need to finish this and then update the schedule; **rest is done**)

Entity:

UserBean

BillBean

Session:

UserSession.

1 Initial planning

1.1 Forming project concept

1.2 Design use cases

1.3 Decide Component framework

2 Setup development environment

2.1 Setup version control

2.2 Setup domain name

2.3 Setup Netbeans and glassFish

3 Get acquainted with EJB

3.1 Reseach EJB components

3.2 Write EJB Toy system

4 Application design

4.1 Data model(UserBean, BillBean)

4.1.1 User/Administrator account(tables)

4.1.2 Friend lists(tables and relationships with users)

4.1.3 Bills and records(tables)

4.2 System infrastructure(UserSession)

4.2.1 User messaging with server(message format)

4.2.2 Server state management(state for every user)

- 4.2.3 Server responses users
- 4.2.2 Security concerns
- 4.2.3 Fault tolerance concerns

5 Implementation

5.1 Interface with email

- 5.1.1 Figure out EJB email receiving/sending
- 5.1.2 Implementation of interface with email

5.2 User account System

- 5.2.1 New account validation and processing system
- 5.2.2 Processing of changing password
- 5.2.3 Setting whether to subscribe from the system

5.3 Billing system

- 5.3.1 Request a bill
- 5.3.2 Accept/Reject a bill and update record
- 5.3.3 Settling a bill
- 5.3.4 Ask for bill report

5.4 Web interface

- 5.4.1 Login/Change password
- 5.4.2 Change setting
- 5.4.3 Show friend list
- 5.4.4 Request/Reject
- 5.4.4 Search/delete bills in certain period

6 Integration and Test

//following is from the last assignment; use it to make proposed changes to the section.

|

v

1) Set Up Version Control

We are using Google code for this part. A repository has already been set.

2) Set up domain name

Michael Glass already has a domain name set for the assignment (mycheapfriend.com)

3) Initial planning

a) Write Project Concept

It was already submitted.

b) Write 1st Iteration Plan

It is currently in progress. All members are working on it.

4) Get acquainted with EJB

a) Write EJB Toy

We will write a very basic Toy system. The system will show minimal message communication between two users. One pair will work on this system while the other comes up with the interface skeleton for our real system.

b) Explain Toy to rest of users (pair-to-pair)

After we have a working toy system and the interface ready, the two pairs will educate each other on the work they have done.

5) Interfacing and testing

a) Write interface & tests for message parser / from/t...

Some of this will be done, when the other pair is implementing the toy system.

b) Backend interfaces / tests

i) Write interface & tests for User object

ii) Write interface & tests for User's nicknames

iii) Write interface & tests for backend object

The work will be divided among the team, the schedule section of this report shows how it will be done.

6) First Iteration 1st project report

The team will write the report for the work done so far.

7) Implementation

a) Implement parser

b) Implement user object

c) Implement user's nicknames

d) Implement backend (bill object)

Again the work is going to split and the actual functionality will be built to complete the system proposed.

8) First Iteration 2nd Project report

The 2nd report for the 1st iteration will be done by the entire team.

9) Figure out emailing

- a) Figure out EJB email sending { specifically to SMS email addresses,
i.e., 6462294050@vtext.com }
- b) Figure out EJB email receiving / parsing
- c) Integrate with backend

At this point, we have the system setup to work with the text messages, now we want to include the e-mailing functionality to the system.

10) First Iteration Final Report

The team will write the final report for the first iteration. The team will also prepare for the demo for the 1st iteration.

11) Code Inspections

Get the code ready to be inspected for the teaching staff.

12) 2nd Iteration Plan

The second iteration starts here.

13) Refactoring

We might want to do the code refactoring, for simplicity and removal of redundancy. Here, we will also need to change the security implementation for the system to make it more secure.

14) 2nd Iteration Progress Report

Write the 2nd iteration progress report.

15) Deploy to internet

Now that the system is ready, we can deploy it to the net, for actual usage.

16) JSP / HTML Front end

- a) Login user html front-end
- b) Bill history html front-end
- c) New bill html frontend

Now that we have the text message and email capability working, for the ease of usability, we can have a web based front end, that might provide a bit extra power to the users, and some extra functionality, regarding their accounts.

17) 2nd Iteration Demos

Get ready for the final presentation.

18) 2nd Iteration Final Report

Write the final report that might include all the work done.

4. Component Framework

Everything is still the same. We are using EJB with NetBeans. It is coupled with the Glassfish server version 2.x.

5. Revised Schedule

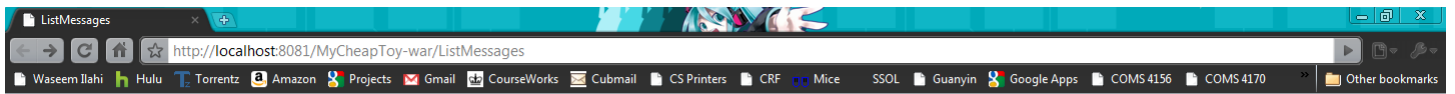
The revised schedule is attached at the end of this document.

6. Toy System Implementation

We have implemented the toy system, using the tutorial presented in the sample first progress report, from last year. There was one difference; we use the tutorial for NetBeans 6.x rather than 5.5. The tutorial is available at the following link: <http://www.netbeans.org/kb/60/ejb30.html>.

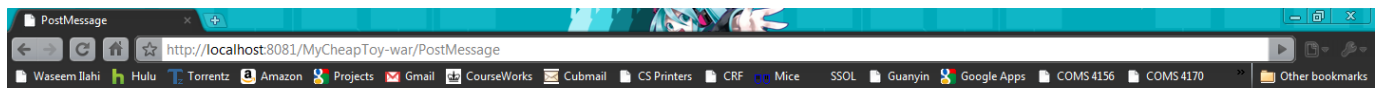
This tutorial solved many problems we were facing. The system implements a web based messaging (blog like) system. The messages posted are shown in a sequence from the first posted to the last. To achieve this functionality, the system creates servlets on the web module and creates the beans on the application server. It uses the entity beans to store the messages to the Glassfish's database. Then it uses the session beans to facilitate the communication between the web interface and the DB server (JMS and Façade). Finally, it uses the message beans to make the two way communication possible (to and from the server).

Implementing this system helped us a lot in building the basic skeleton of our real system. We are employing the same messaging scheme, to create user entities (persistent) on the server and then access the entries in the table. We have included the screen shots from one of the runs of the toy system that show its working.



Message List

[Add new message](#)

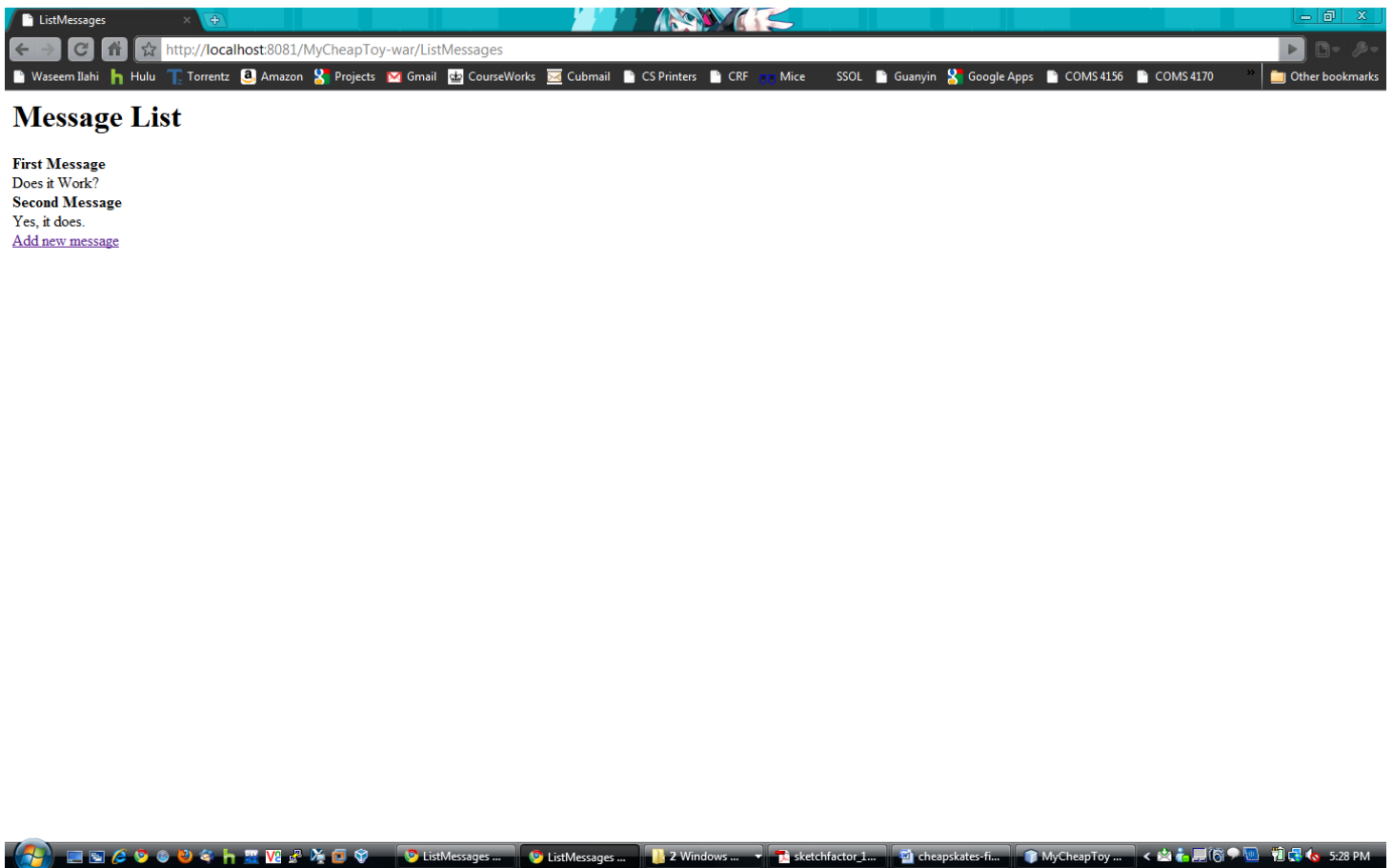


Type Message Name And Text

Title:

Message:





7. Controversies

There are no controversies among the team at this point.