*Splitwise++*

Prateek Pisat

Computer Science

Northeastern University

Boston, United States

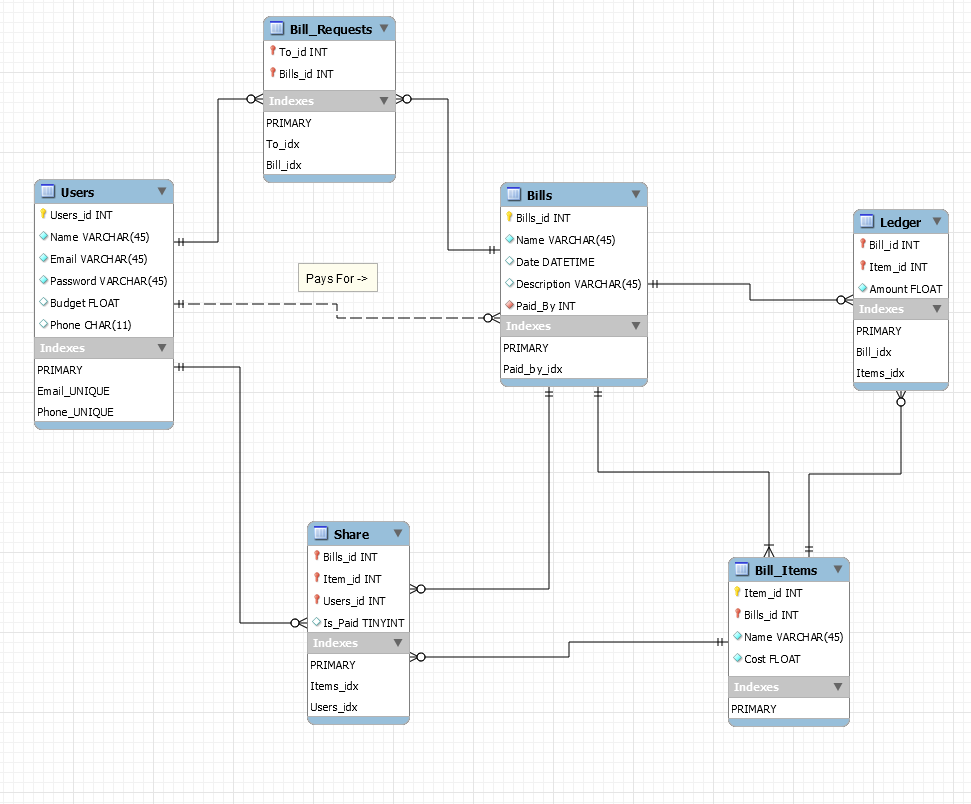
pisat.p@husky.neu.edu

*Abstract* — Splitwise is a free tool for friends and roommates to track bills and other shared expenses. Although the app is incredibly useful, it has few bugs that make the app unreliable. And hence, I intend to implement an improved version of the app. Essentially the app will be work exactly as Splitwise with common and annoying bugs fixed and some added features.

# Problem Statement

The app primarily consists of Users and Bills. Each user needs to register before they can use the app’s features. Each user is associated with a Name, Email, Password, Phone Number and a Budget. Registered users can add Bills. Each bill is associated with a Name, an optional Description, a date, and the user who paid the bill. Each bill consists of Items. Each item belongs to a bill and is associated with the name of the item and cost. After adding a bill and items to that Bill the user will have identify, which other users will they be sharing the bill with. So, once a complete bill is created the creator must send requests to other users, of the creditors choice, to participate in the bill, so as to identify what items do they want, and should be charged appropriately.

# E.R Diagram



1. Table Descriptions

Table Descriptions:

1. Users: All the registered users will be stored in this table.

a. Primary Key: Users\_id

2. Bills: All the bills generated by any user will be stored in this table

a. Primary Key: Bills\_id

b. Foreign Key: Paid\_By References Users(Users\_id)

3. Bill\_Items: The Item in each bill will be added here, referenced by its bill id.

a. Primary Key: Item\_id + Bills\_id

b. Foreign Key: Bills\_id References Bills(Bills\_id)

4. Share: This table maintains records which user is sharing which item.

a. Primary Key: Users\_id + Item\_id + Bills\_id

b. Foreign Key: Users\_id References Users(Users\_id), Item\_id References Bill\_Items(Item\_id), Bills\_id References Bills(Bills\_id)

5. Ledger: This tables stores the shared cost of an item, after it is divided amongst all the people who are sharing it.

a. Primary Key: Bill\_id + Item\_id

b. Foreign Key: Bill\_id References Bills(Bills\_id), Item\_id References Bill\_Items(Item\_id)

6. Bill\_Requests: This tables stores all the requests the creditor sends to the users who they think should be sharing in the bill.

a. Primary Key: To\_id + Bills\_id

b. Foreign Key: To\_id References Users(To\_id), Bills\_id References Bills(Bills\_id)

# Introduction

Problems with Splitwise:

• The balance bug: The app has this weird bug, where it fails to calculate the correct balance in certain situations. Consider x, y to be two users. Let x owe y some money (or vice-versa). Let this transaction be a part of some group g. Let y owe x some money (or vice-versa), however this transaction is not part of any group. In this case the app incorrectly calculates the balance x owes y. However, this case is not common, and may or may not occur always. This leads me to believe that, the bug may occur due to a random crash OR there might be some over-complication in how the transactions are represented.

• Participation Problems: For any set of items brought, it is not the case that everyone necessarily wants each item. Certain people may want certain items, others may not. The app however, assumes that all the people (in a group/ household) want each of the items brought. So, users need to manually (not a part of the app) specify what items they want to be included in. This leads to one bill being split into various sets of items, depending on what items, of one bill, people want to be included in. This leads to a huge added overhead, specially, if the bill contains a large number of items.

• No Concrete Record of Bills: Another problem with the above-mentioned complication is, since one bill is divided into separate, independent and disconnected pieces, there is no reference to the entire bill, with all the items, only a list of micro-transactions, that if joined together, will form the complete bill. However, the App UI doesn’t provide any such features.

I intend to fix these common issues/ bugs. Along with fixing these problems I have added a few new features like a budget tracking system.

Each user will be allowed to set a monthly budget. The app will keep track of how close the user is to achieving that budget. If the user’s expenses surpass the set budget the app will identify certain expensive items that the user can avoid the next month.

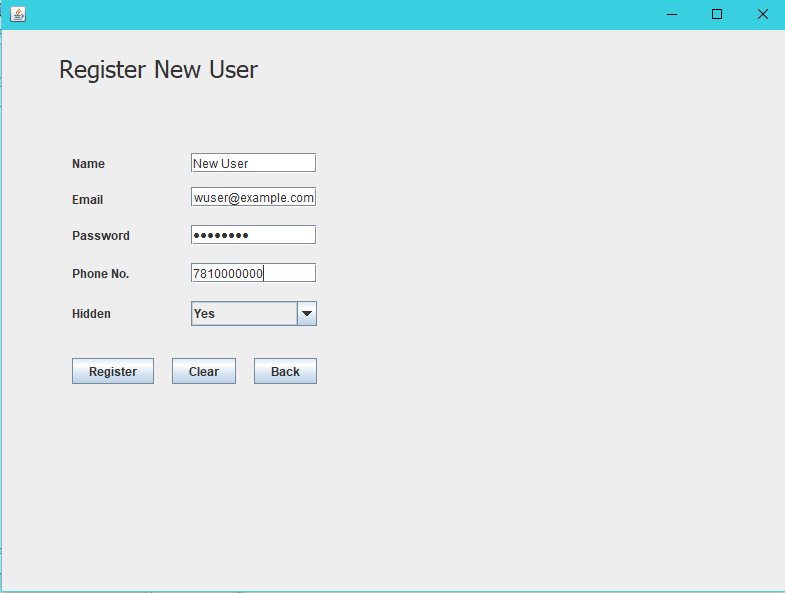
# The Components/ Modules

The app uses the following modules/ components:

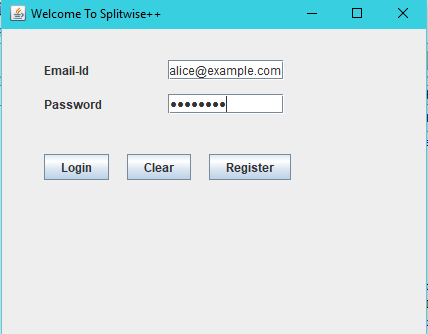
1. Login/ Registration:

To use the app all users, need to register. The registration form screen-shot describes all the information a user need to provide at the time of registration. NOTE: The Hidden section is added on purpose, however is not used in the scope of this project, and will be used to implement addition features, such as profile privacy.

Once a user has registered, they can proceed to login to the app and start using its features. The login screen’s screen-shot is provided below.



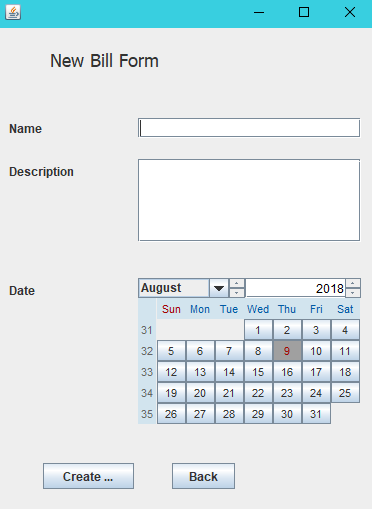
*Fig.1. Registration*



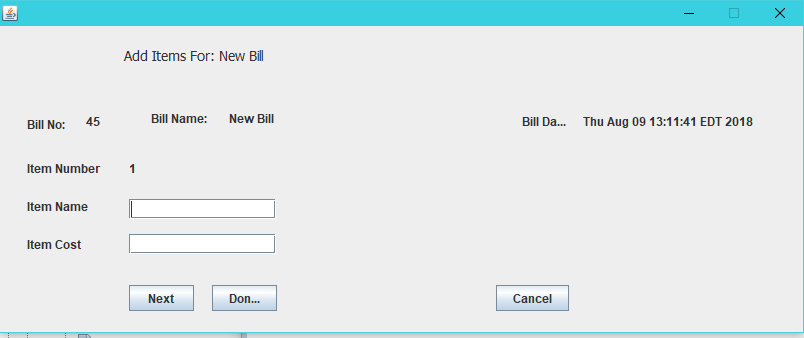
*Fig.2. Login*

1. Adding a New Bill:

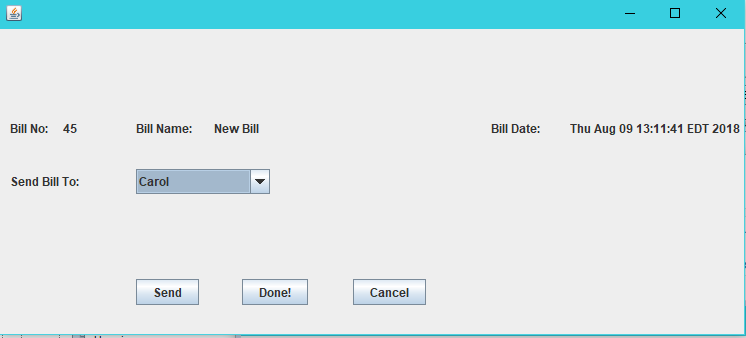
The crux of the app is to allow users to add new bill and share these bill with other users to appropriately split the said bill. Creating a new bill is a multi-step procedure, to satisfy the database constraints. Creating a bill is divided into 3 steps, adding a bill, adding items to that bill and sending the bill to the users the creator wishes to share the bill with. A new Bill is identified by its name, a description and the date. To add an item to a bill, mention the name and cost of the said item, then click “next”. After adding the last item click “done”. To send the bill to a specific person, select the person from the drop-down menu, and then click “send”. After sending to the last person click “done”. This is explained in the following screen-shots.



*Fig.3. New Bill*

**

*Fig.4. Add Items*

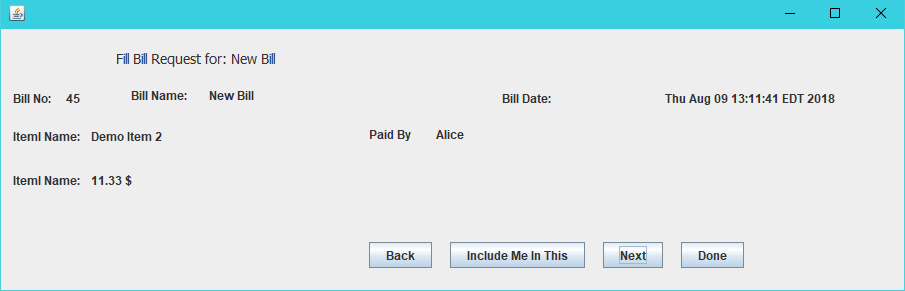
**

*Fig.5. Send Bill*

1. Fill a Bill Request:

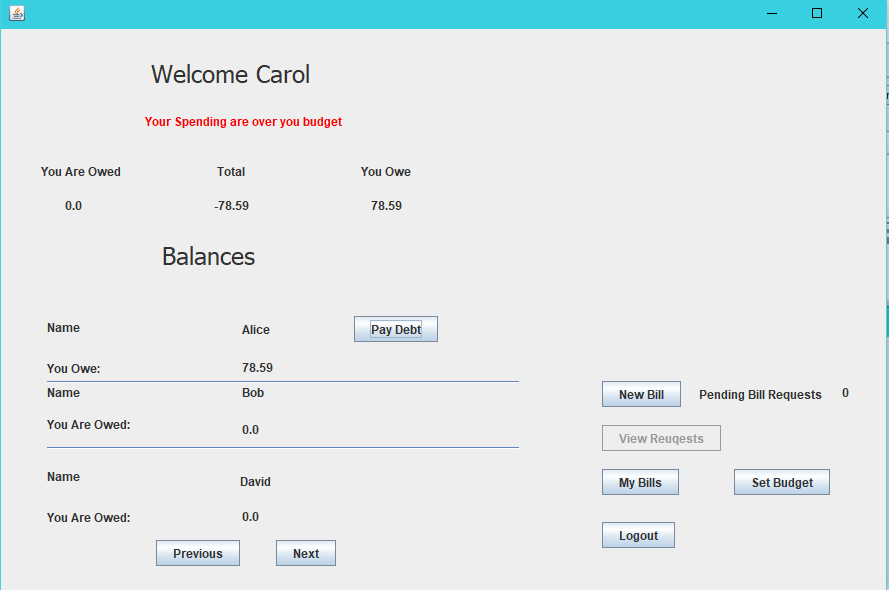
If the user has certain bills sent to them, then the “Pending Bill Requests” label will indicate the number of bill requests the person has yet to fill. To fill a bill-request the user need to click on “View Requests” on the home screen. This will show all the pending bill requests. If the user chooses to participate in a bill, they should click on “Participate”, else “delete” to delete the bill-request.

If the user chooses to participate in the bill, then the following screen will load, which will display all the item in that bill, allowing the user to choose, which item they wish to pay for.



*Fig.6. Item Selection*

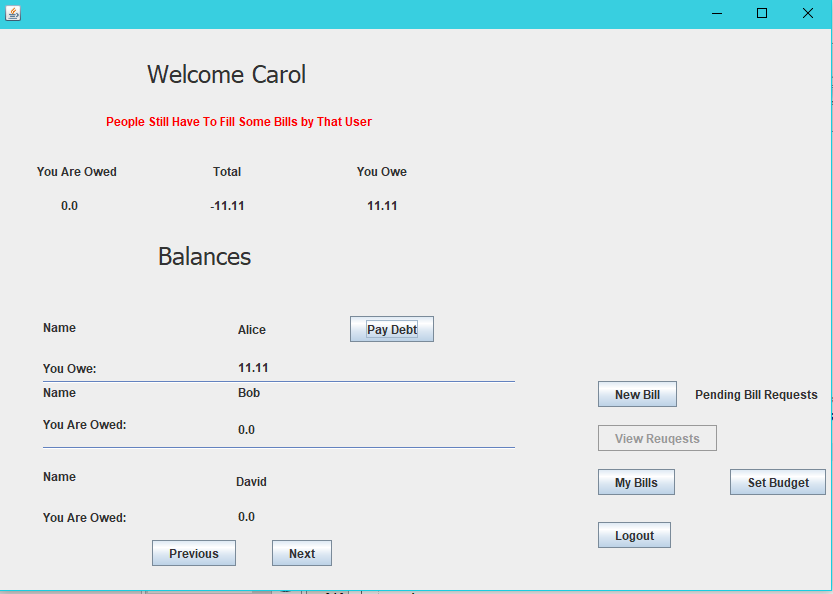
After the user is done selecting what items they wish to pay for they must click “Done”, this will load the home screen, with the update balance, i.e. how much they owe someone or how much someone owe them. Consider the following screen-shot as an example:



*Fig.7. User Homepage.*

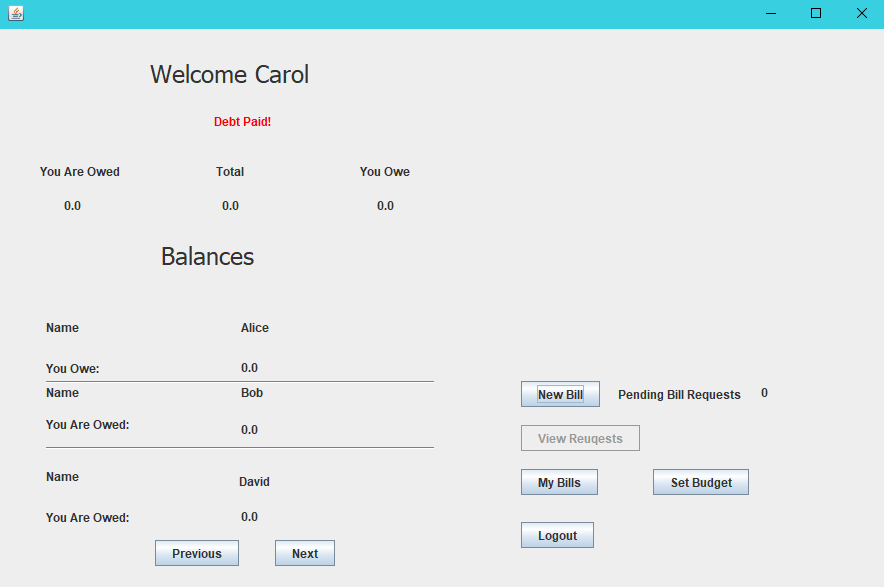
1. Paying a Debt:

The user may choose to pay the debt they owe to some other user; however, they can’t pay unless all the bill requests sent by the creator of the bill have be filled by all the user to who the request was sent to. This is shown in Fig. 8. This ensures that the cost of the items is distributed fairly, and person paying first doesn’t have to pay more than the person paying later.

**

*Fig.8. Requests are remaining.*

Once all the bill-requests have been filled, a debtor can then click the “pay-debt” button and debt will be cleared. If A owes B some money, and B owes A some money, whoever owes more will see the “Pay Debt” button enabled and the amount that they owe to the other user, on clicking the button, all the debts amongst A and B will be settled. This can be seen in Fig.7 and Fig 9.

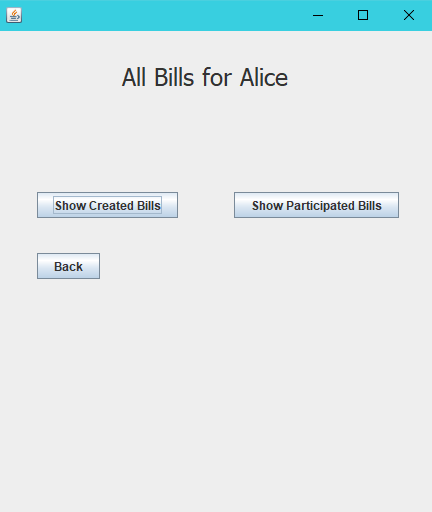


*Fig. 9. After Paying a Debt*

As shown in the screen-shot, if the user’s spending goes above their set budget, then they get a message indicating that they are spending more than what they wish to. Setting and updating a budget is explained in detail in a later sub section.

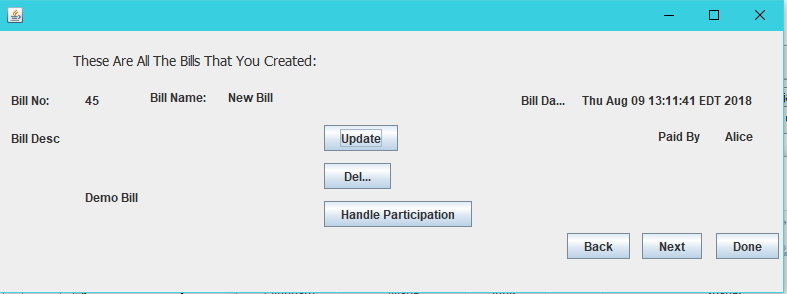
1. Managing Bills

Bills are primarily divided into two categories; the bills that a user created, and the user participated in. All these bills can be view or edited by clicking on “My Bills” on the user homepage.

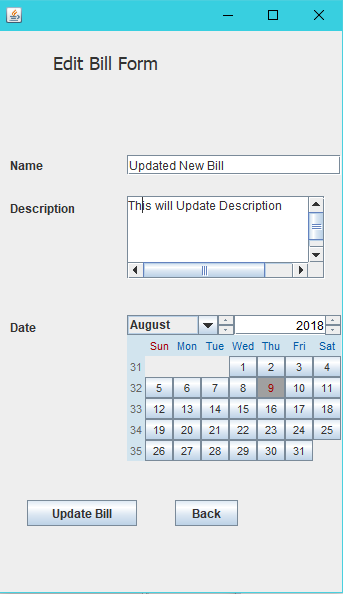


*Fig.10. My Bills*

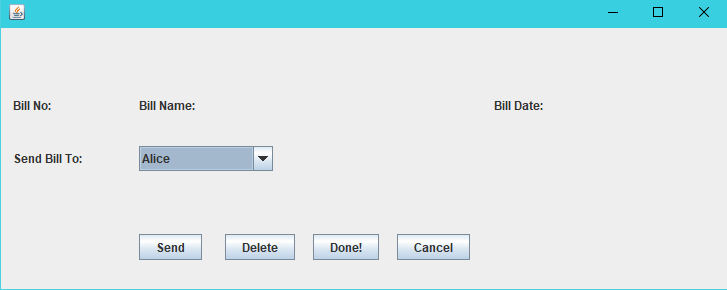
Then the user has the option of viewing/ editing the bill that they created, or the bill that they participated. Either of which can be done clicking on the appropriate button. If the user selects “Created Bills” then they can update the name, date, description of the bill and update the user with who the bill was shared with. If the user selects “Participated Bill”, then the user can update their participation in that bill. Examples are shown in Fig. 11 through Fig. 15.



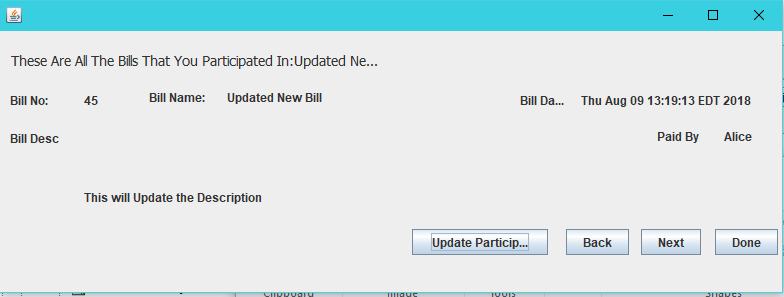
*Fig.11. View/ Update Created Bills.*

**

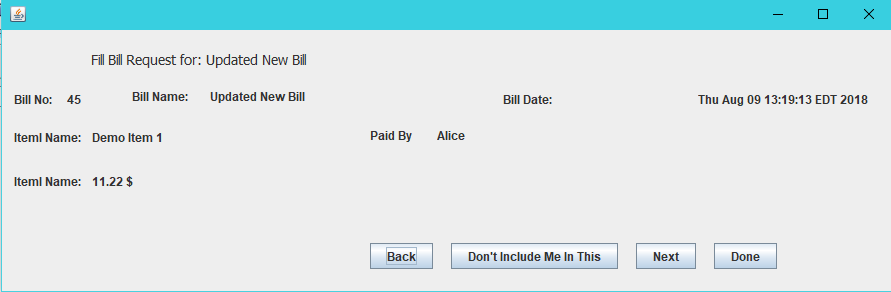
*Fig.12. Update Bill.*

**

*Fig.13. Update Bill Participation (The users the bill was shared with)*



*Fig.14. View/ Update Participation*

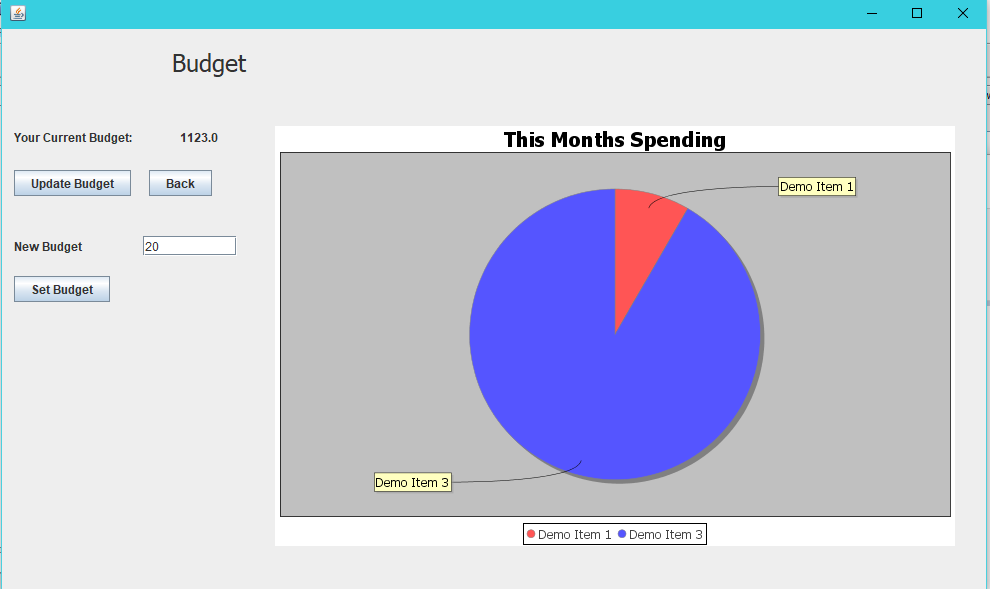
**

*Fig.15. Update Item Participation.*

1. Setting a Budget:

Each user can set a budget. If the spending if the user, exceed the budget then user home page will display a message indicating that user’s spending is more than what they intended to. This is clearly show in Fig.8.

To set or update the budget, the user need to click on the “Set Budget” button on the home page. The user will see the following screen:



*Fig.16. User Budget.*

Here the user can set or update their previously set budget. Additionally, a Pie-Chart of the user’s spending for that month, is show to the right. This provides a quick way for the user to identify what item are they spending more on.

# Use cases

As mentioned in the description, this app/ Project is for friends/ roommates to keep track of expenses, and ensure that every bill is split equally, and everyone gets paid.

Additionally, this app can also be used for maintaining a budget, to keep track of the money spent that month, and what items was the money spent on.

The app can also be used as a general record of all the bills that we generated by a user.

##### Technical Specifications

The Technical Specifications under which the app was developed:

* DBMS: MySQL. Version 8.0
* Backend: Java. Version 1.7+
* Frontend: Java Swing Applet.
* Libraries:
  + JFreeChart for Data Visualization, Version 1.0.19
  + Connector/J for Database connection, Version 8.0.12
  + JCalender for a GUI calendar, Version 1.4

##### Conclusion

The app successfully fixes all the bugs and errors previously mentioned. Users can now select the individual items that they would want to pay for and are hence charged appropriately. Since, the bill is no longer divided into records of individual items, a complete record of all the bill created is stored by the app and allows the user to view the bills they have participated in and additionally edit the bills that they have created.

The simple database design eliminates the ‘balance bug’ and proper indexing helps in improved performance. The Budget Tracking feature helps users to identify when their monthly spending is above their predefined budget. This also, shows a user Pie-Chart of how the user has been spending their money, to help them identify items that they could avoid.

##### References

These are the research papers and websites that we used as a reference.

1. <https://dev.mysql.com/doc/>
2. <https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/>
3. <http://www.jfree.org/jfreechart/api/javadoc/index.html>
4. <https://secure.splitwise.com>