

# PROJECT REPORT

OF

“ SPLIT PAY ”



**Delhi Technological University**

Submitted in the partial fulfillment of the Degree of bachelor of Technology  
In  
Software Engineering

**SUBMITTED BY :-**  
Shubham (2K19/SE/119)

**GUIDED BY :-**  
Mr. Sanjay Kumar

# TOPIC OF THE PROJECT

SPLIT PAY - Split Bill Among Friends In Mesh Of Groups



Subject - DATA STRUCTURES

Professor In-charge - Mr. Sanjay Kumar

**NOVEMBER, 2020**

# DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

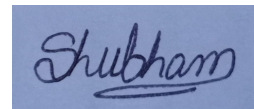
Bawana Road, Delhi - 110042

## **CANDIDATE'S DECLARATION**

I, (Shubham (2K19/SE/119)) student of B.Tech (Software Engineering), hereby declare that the project dissertation titled, “ Split Pay ” which is submitted by me to Department of Software Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of degree of Bachelor of Technology, is original and not copied from any source without citation. This work has not previously formed basis for award of any degree, diploma associateship, fellowship or any other similar title or recognition.

Date : November 18, 2020

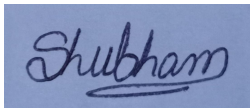
Place : Delhi Technological University, Delhi



Shubham (2K19/SE/119)

## ACKNOWLEDGEMENT

I, (Shubham (2K19/SE/119)) express our sincere gratitude to Mr. Sanjay Kumar for his valuable guidance and timely suggestions during the entire duration of our dissertation work, without which this work would have been possible. I would like to convey my deep regards to all faculty members of Department of Software Engineering, who have bestowed their great efforts and guidance at appropriate time without which it would have been very difficult on our part to finish this work. I would like to thank my friends for their advice and pointing out the mistakes.

A blue rectangular box containing a handwritten signature in dark blue ink. The signature is written in a cursive style and reads "Shubham".

Shubham (2K19/SE/119)

## **ABSTRACT**

We often go on trips and movies with friends, we end up paying for each other at different points of our trips and usually end up doing mental calculations of who paid what and then sharing the expense. There are few web applications that lets room-mates, friends track personal finances and share expenses they made over time, but we make most of the expenses when we are outside on the go and we don't have a laptop or solid internet connection to update the expenses made. This android application helps those people to calculate expenses and debt of each persons in the group. And also our application is run in offline mode as well. We feel that this app is very useful to manage these small personal finance issues and keep everyone in the loop.

# **INDEX**

1. Objective
2. Introduction
3. Strength and Weakness
4. Data Structures that are used
5. System Design and Algorithm
6. Result and Analysis
7. Conclusion
8. References
9. Annexure (Code)

## **OBJECTIVE**

To develop an application in which a user is able to track his expenditure and also how much he/she has to pay or to be paid. Our application will be used to split bill among all the friends and colleagues present in the group. Make this application able to run in offline mode as well.

## **INTRODUCTION**

The group expense tracker or split pay manages all your financial accounts in one convenient place and makes your money management much easier and handy. Our expense tracking software lets you compare your spending with other other friends who share similar group circles so you can see how much they typically spend and identify the best ways to save on your money. With expense tracker / split pay you can view your investments at each stage and know that can you expend your money further or not. This application not only calculate the expenses of whole group but can also calculate the expenses between two individual persons and the expenses between some people in the group. Our application is new concept very easy and simple to handle, not very complicated as other mobile applications. The main benefit of using mobile phone is that it's simple and innovative.

## **STRENGTHS AND WEAKNESS**

### **Strengths :-**

1. It is a mobile based application.
2. It can be run in offline mode as well.
3. The application will not only work on android based mobile handset but also on web.
4. It is able to track expenses of whole group.
5. It is able to track expenses of some people in the group.
6. It is able to track expenses of each and every individual person present in the group.
7. Generate the group report which will contain expense details of each member in a group.

### **Weakness :-**

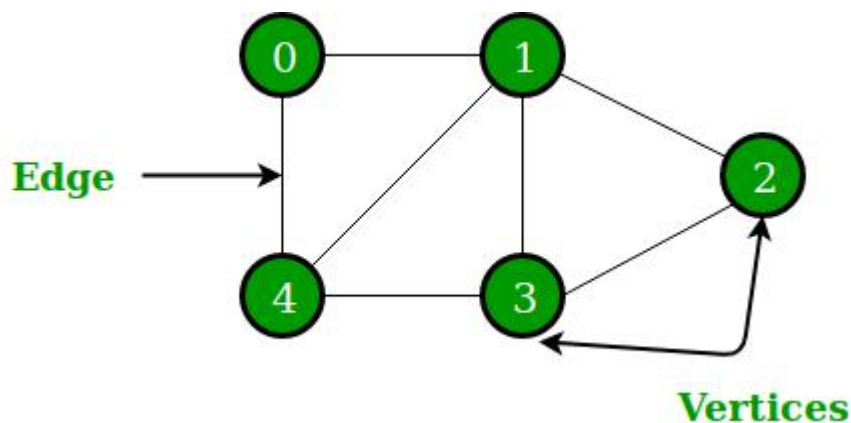
1. A group should have a common currency.
2. User has to enter data each time to see his/her expenses.



## DATA STRUCTURES THAT ARE USED

### 1. GRAPHS :-

A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph can be defined as, *A Graph consists of a finite set of vertices(or nodes) and set of Edges which connect a pair of nodes.*

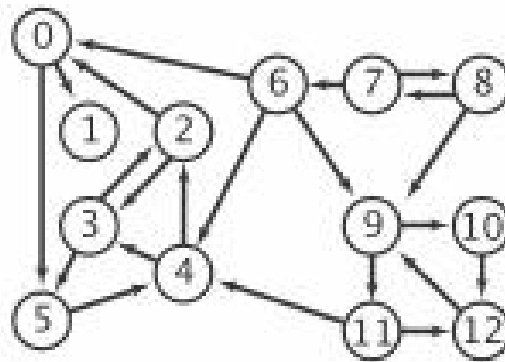


In the above Graph, the set of vertices  $V = \{0,1,2,3,4\}$  and the set of edges  $E = \{01, 12, 23, 34, 04, 14, 13\}$ .

Graphs are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. In our system graph to link all the people in the group in which each individual person show the node and if their exist any payment between two friends then there is edge between them.

## Directed Graph :-

A *directed graph* (or *digraph*) is a set of *vertices* and a collection of *directed edges* that each connects an ordered pair of vertices. We say that a directed edge *points from* the first vertex in the pair and *points to* the second vertex in the pair. We use the names 0 through V-1 for the vertices in a V-vertex graph.



In our system, the edge is from the person that have to give money to the person who want money from that person.

## Weighted Graph :-

A weighted graph is a graph in which each branch is given a numerical weight. A weighted graph is therefore a special type of labeled graph in which the labels are numbers (which are usually taken to be positive). In our system the weight represents the debt amount of each person.

## 2. HASHMAP :-

A HashMap is a data structure that is able to map certain keys to certain

values. The keys and values could be anything. For example, if I were making a game, I might link every username to a friends list, represented by a List of Strings.

HashMaps are much faster for retrieving data than arrays and linked lists. A sorted array could find a particular value in  $O(\log n)$  with binary search. However, a HashMap can check if it contains a particular key in  $O(1)$ . All keys must be unique.

HashMaps use an array in the background. Each element in the array is another data structure (usually a linked list or binary search tree). The HashMap uses a function on the key to determine where to place the key's value in the array.

In our system we use two inbuilt unordered map, named ourMap and userMap. In ourMap the key is of type int and value is of type string and in userMap the key is of type string and value is of type int.

ourMap is store key from 0 to n-1 and value as name of the persons present in group. userMap is store key as name of the persons present in group and value from 0 to n-1, where n is the total number of people present in the group.

### **DYNAMIC ARRAY :-**

It is also known as growable array, resizable array, dynamic table, mutable array, or array list. It is a random access, variable-size list data structure that allows elements to be added or removed. It is supplied with standard libraries in many modern mainstream programming languages. Dynamic arrays overcome a limit of static arrays, which have a fixed capacity that needs to be specified at allocation.

A dynamic array is not the same thing as a dynamically array, which is an array whose size is fixed when the array is allocated, although a dynamic array may use such a fixed-size array as a back end.

In our system, we uses a dynamic 2D array named edges to store the weight of the edge, in short it is used to make adjacency matrix of size  $n \times n$ , where  $n$  is the total number of people present in the group.

## **SYSTEM DESIGN**

Now we began with the design phase of the system. System design is a solution, a “ HOW TO ” approach to the creation of a new system. It translates system requirements into ways by which they can be made operational. It is a translational from a user oriented document to a document oriented programmers. For that, it provides the understanding and procedural details necessary for the implementation. Here we use Algorithms to supplement the working of the new system. The system thus made should be reliable, durable and above all should have least possible maintenance costs. It should overcome all the drawbacks of the old existing system and most important of all meet the user requirements.

### **ALGORITHM :-**

1. Take input of the group name as string.
2. Take input number of people present in the group as integer  $n$ .

3. If user enter number of people present in the group less than one then release error message.
4. Ask user to enter the names of the people present in that group store name as string.
5. Declare two inbuilt unordered map, let's ourMap and userMap. In ourMap the key is of type int and value is of type string and in userMap the key is of type string and value is of type int.
6. ourMap is store key from 0 to n-1 and value as name of the persons present in group. userMap is store key as name of the persons present in group and value from 0 to n-1.
7. Now make adjacency matrix which stores edges in between the group peoples.
8. The graph formed is directed and weighted.
9. The due amount of each friend show the weight of the edge.
10. Now call addExpense(), repayDebt() and currentPosition() functions according to user need.

#### **Algorithm of addExpense function :-**

1. This function add the expense to whole of the group or add expense to some people of the group.
2. Ask user to enter the expense amount and which member of the group have paid that amount.
3. If the amount that enter is less than zero than release the error message.
4. Ask user the expense amount is used by whole the group or it is used by some people of the group.
5. If the amount is used by whole of the group then divide the amount by total number of people present in the group and update the adjacency matrix for each member.
6. If the amount is used by some people of the group then divide the amount by number of people present that uses amount and update the adjacency matrix for their respective positions.

**Algorithm of repayDebt function :-**

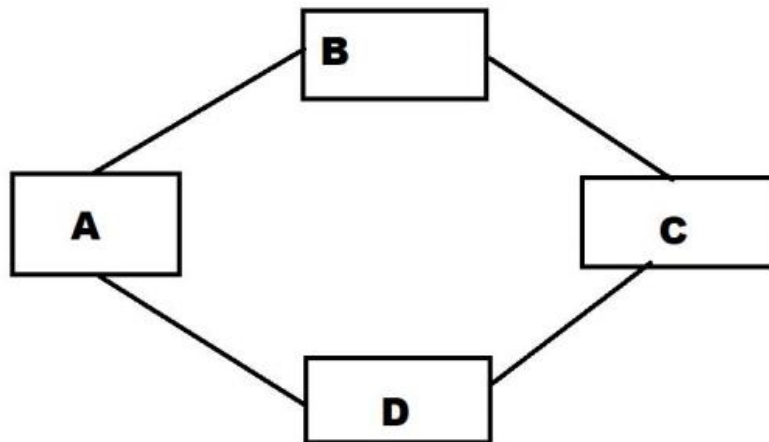
1. This function minimize the debt expense of those peoples who have repay their debt.
2. Ask user to enter the debt amount which is paid and the name who paid.
3. If the amount that enter is less than zero than release the error message.
4. Then ask user to enter the name who received that amount.
5. Now update the adjacency matrix at respective position of people who paid and who received the amount.
6. Continue step 5 till all the debts complete or when user want to stop.

**Algorithm of currentPosition function :-**

1. This function generate the report of all expenses of group and show which person has to complete his/her expenses and which person can get that from expense amount.
2. Traverse whole the adjacency matrix and generate the report.

## RESULT AND ANALYSIS

Let's start with an example suppose there is a group of four colleagues namely A, B, C, D.



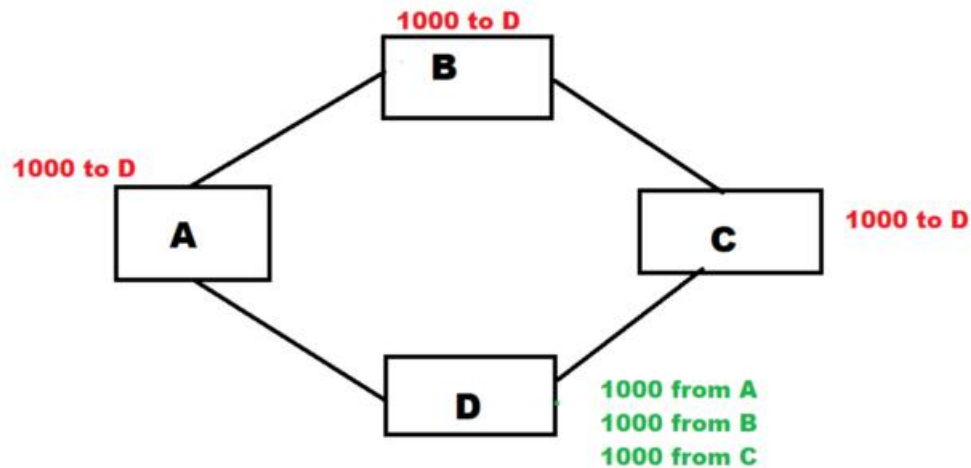
```
Enter the group name : ABCD
Enter number of people in group ABCD : 4
Enter the names of people in group ABCD :-
A
B
C
D

***** MAIN MENU *****

Enter 1 : For add expense to group ABCD
Enter 2 : If anyone wants to repay his/her debt.
Enter 3 : For check current position of debt of group ABCD
Enter -1 : For exit.

Enter function that you want to perform :
```

They went to a restaurant to eat, the bill amount was 4000.  
'D' paid the full bill now others have to pay to 'D' their share of money.  
So, A, B, C will pay 1000 rupees each to 'D'.



```
Enter function that you want to perform : 1
Enter -1, if there is no more expenses.

Enter the Expenses to group ABCD : 4000
Who Paid : D
Enter 1, for add expense to whole group ABCD
Enter 2, for add expense to some people of ABCD
1

Enter the Expenses to group ABCD : -1
*****

***** MAIN MENU *****

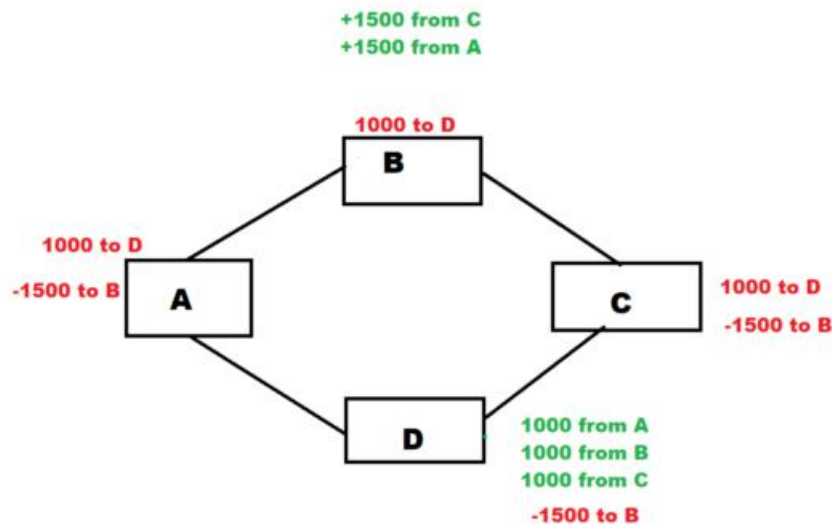
Enter 1 : For add expense to group ABCD
Enter 2 : If anyone wants to repay his/her debt.
Enter 3 : For check current position of debt of group ABCD
Enter -1 : For exit.

Enter function that you want to perform : 3
D wants 1000 rupees from A
D wants 1000 rupees from B
D wants 1000 rupees from C
*****
```



Let's say they decided to pay to ' D ' later on some other day.

Again these four colleagues went to a restaurant, this time the total bill was 6000 rupees and this time B paid the full bill. Now the question arises that how much should each one should pay to one another so that everyone has paid their share.



```
Enter function that you want to perform : 1
Enter -1, if there is no more expenses.

Enter the Expenses to group ABCD : 6000
Who Paid : B
Enter 1, for add expense to whole group ABCD
Enter 2, for add expense to some people of ABCD
1

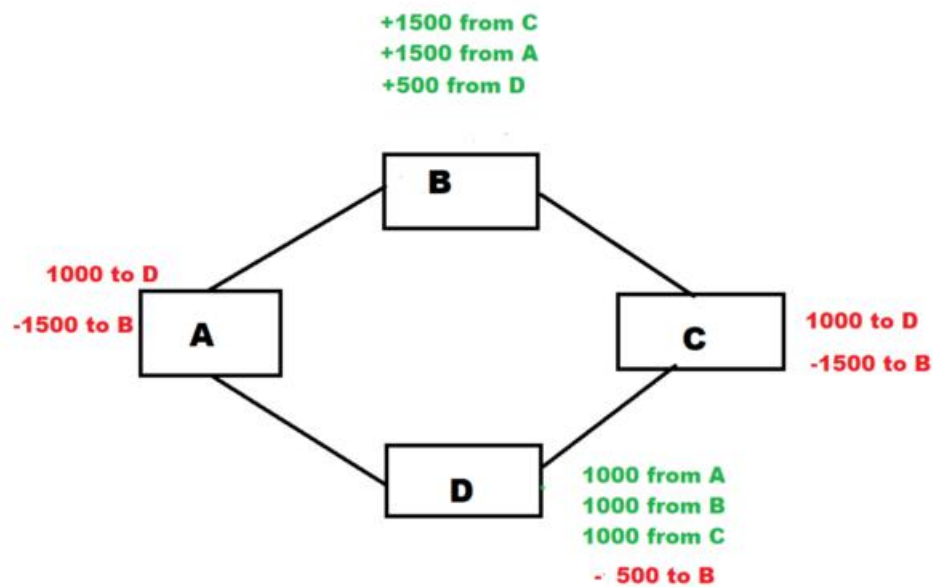
Enter the Expenses to group ABCD : -1
*****

***** MAIN MENU *****

Enter 1 : For add expense to group ABCD
Enter 2 : If anyone wants to repay his/her debt.
Enter 3 : For check current position of debt of group ABCD
Enter -1 : For exit.

Enter function that you want to perform : 3
B wants 1500 rupees from A
D wants 1000 rupees from A
B wants 1500 rupees from C
D wants 1000 rupees from C
B wants 500 rupees from D
*****
```

Net Balance :



```
Enter function that you want to perform : 3
```

```
B wants 1500 rupees from A
```

```
D wants 1000 rupees from A
```

```
B wants 1500 rupees from C
```

```
D wants 1000 rupees from C
```

```
B wants 500 rupees from D
```

```
*****
```

```
***** MAIN MENU *****
```

```
Enter 1 : For add expense to group ABCD
```

```
Enter 2 : If anyone wants to repay his/her debt.
```

```
Enter 3 : For check current position of debt of group ABCD
```

```
Enter -1 : For exit.
```

```
Enter function that you want to perform : -1
```

```
You are Exit.
```

```
Process returned 0 (0x0) execution time : 628.409 s
```

```
Press any key to continue.
```

- ~ 'A' has to pay 1500 rs. to 'B' and 1000 rs. to 'D'.
- ~ 'B' has to get paid 1500 rs. from 'A', 1500 rs. from 'C' and 500 rs. from 'D'.
- ~ 'C' has to pay 1500 rs. to 'B' and 1000 rs. to 'D'.
- ~ 'D' has to get paid 1000 rs. from 'A' and 1000 rs. from 'C'. And also 'D' has to pay 500 rs. to 'B'.

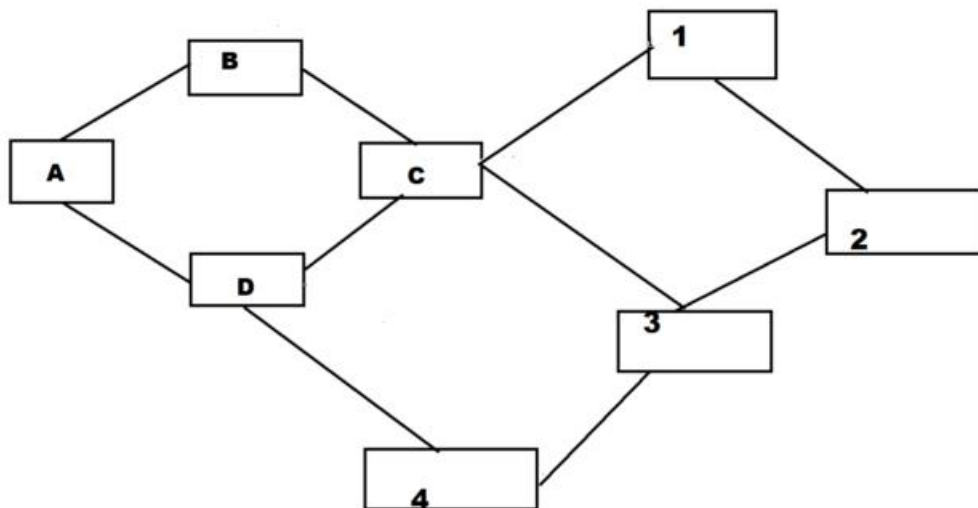
### **Real Case :-**

People in different link :

In reality different people have different friend circles. In that friend circle some friends are common while others are not.

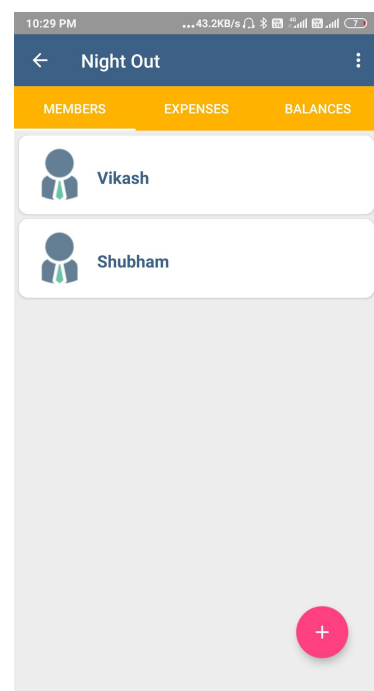
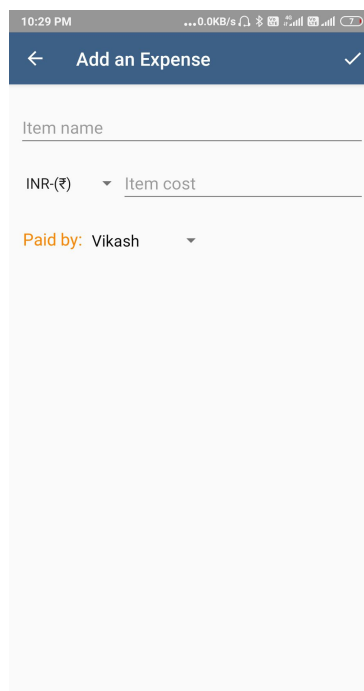
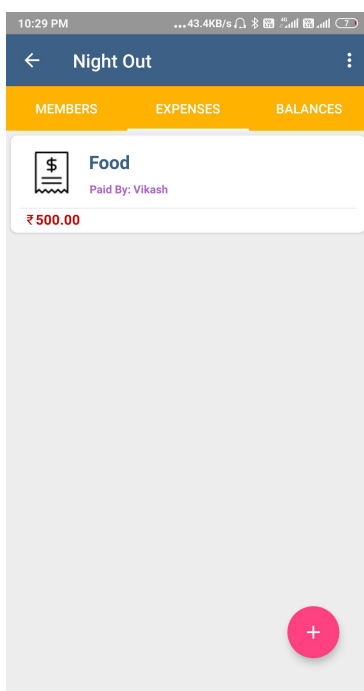
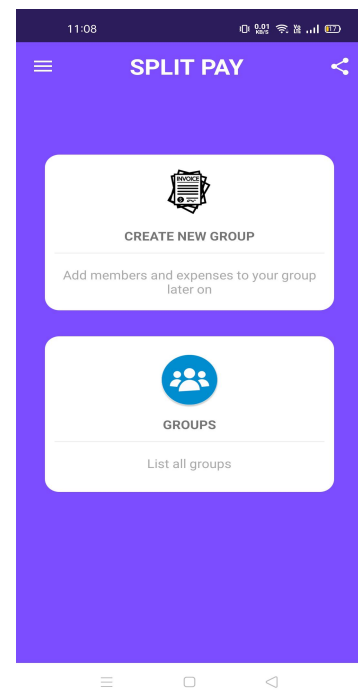
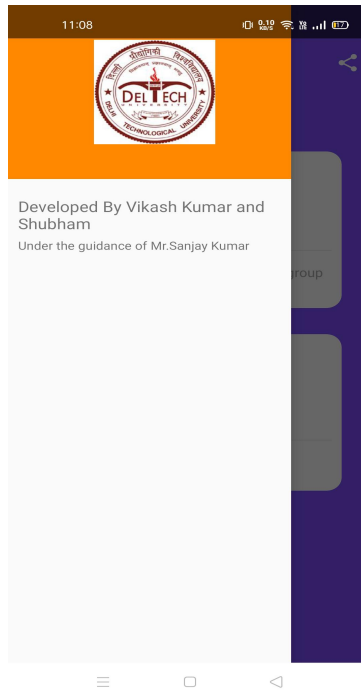
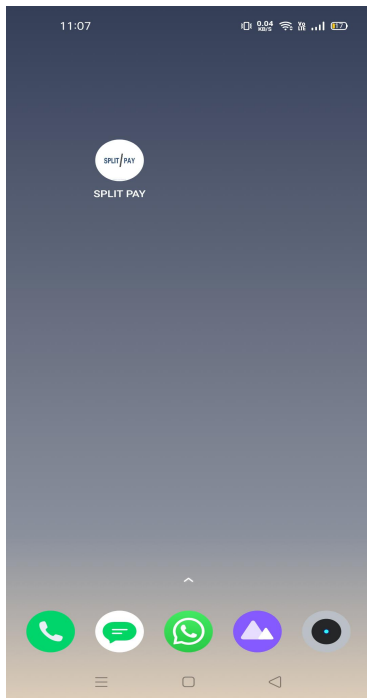
In the following block diagram 'C' and '4' have common friends.

So in reality a complex link is formed between people.



Our project is able to implement these types of situation as well.

## Some snapshot of our application :-



## **CONCLUSION**

In this report have discussed a flexible solution for keeping track of group expenses on the go rather than the contemporary way of making mental and sometimes erroneous calculations. Our applications deals with developing an application to overcome the shortcomings of the existing expense - tracking applications.

We have also made a comparative study of various available technologies to solve and their feasibilities and advantages / disadvantages. Finally we solve many of the problems of the previous build systems and make an Android mobile application synchronized with a web - application to track the expenses. This report discussed the proposed system in detail, with regards to its functionality, strengths and weaknesses.

## **REFERENCES**

- [1]. "Some questions on hashmap data structure", [Online]. Available : <https://leetcode.com/problems/design-hashmap/description/> [Accessed : sep 21, 2020].
- [2]. "An explanation of graph data structure", [Online]. Available : <https://stackoverflow.com/questions/50309576/graph-data-structure-with-complex-edge-data> [Accessed : oct. 15, 2020].
- [3]. "Some questions on graph data structure", [Online]. Available : <https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/> [Accessed : oct 23, 2020].

# **ANNEXURE**

## **CODE OF THE PROJECT**

```
#include <iostream>
#include <string>
#include <unordered_map>
using namespace std;

void repayDebt(double** edges, int n, unordered_map<string, int>&
    userMap, string group)
{
    double amount;
    cout << "Enter -1, when there is no one left for pay debt." << endl;
    cout << "\nHow much amount debt completes : ";
    cin >> amount;

    while(amount != -1)
    {
        if(amount <= 0)
        {
            cout << "\nWrong amount !!" << endl;
            return;
        }

        string nameWhoPaid, nameWhoReceived;
        cout << "Who Paid : ";
        cin >> nameWhoPaid;
        cout << "Who received : ";
        cin >> nameWhoReceived;
```

```

        if(userMap.count(nameWhoPaid) == 0)
        {
            cout << "Person " << nameWhoPaid << " doesn't exist in
group " << group << " !" << endl;
            return;
        }

        if(userMap.count(nameWhoReceived) == 0)
        {
            cout << "Person " << nameWhoReceived << " doesn't exist
in group " << group << " !" << endl;
            return;
        }

        int personWhoPaid = userMap[nameWhoPaid];
        int personWhoReceived = userMap[nameWhoReceived];

        edges[personWhoReceived][personWhoPaid] =
edges[personWhoReceived][personWhoPaid] + amount;

        if(edges[personWhoReceived][personWhoPaid] >
edges[personWhoPaid][personWhoReceived])
        {
            edges[personWhoReceived][personWhoPaid] =
edges[personWhoReceived][personWhoPaid] -
edges[personWhoPaid][personWhoReceived];
            edges[personWhoPaid][personWhoReceived] = 0;
        }
        else if(edges[personWhoReceived][personWhoPaid] ==
edges[personWhoPaid][personWhoReceived])
        {
            edges[personWhoReceived][personWhoPaid] = 0;

```

```

        edges[personWhoPaid][personWhoReceived] = 0;
    }
    else
    {
        edges[personWhoPaid][personWhoReceived] =
edges[personWhoPaid][personWhoReceived] -
edges[personWhoReceived][personWhoPaid];
        edges[personWhoReceived][personWhoPaid] = 0;
    }

    cout << "\nHow much amount debt completes : ";
    cin >> amount;
}
}

void addExpense(double** edges, int n, unordered_map<string, int>&
    userMap, string group)
{
    double expense;
    cout << "Enter -1, if there is no more expenses." << endl;
    cout << "\nEnter the Expenses to group " << group << " : ";
    cin >> expense;

    while(expense != -1)
    {
        if(expense <= 0)
        {
            cout << "Wrong Expense !!" << endl;
            return;
        }

        string nameWhoPaid;
        cout << "Who Paid : ";

```



```

cin >> nameWhoPaid;

if(userMap.count(nameWhoPaid) == 0)
{
    cout << "Person " << nameWhoPaid << " doesn't exist in
group " << group << " !" << endl;
    return;
}

int personWhoPaid = userMap[nameWhoPaid];

int option;
cout << "Enter 1, for add expense to whole group " << group <<
endl;
cout << "Enter 2, for add expense to some people of " << group
<< endl;
cin >> option;

if(option == 1)
{
    for(int i=0; i<n; i++)
    {
        if(i == personWhoPaid)
        {
            continue;
        }

        double expensePerPerson = expense/n;
        edges[i][personWhoPaid] = edges[i][personWhoPaid]
+ expensePerPerson;

        if(edges[i][personWhoPaid] >
edges[personWhoPaid][i])

```

```

        {
            edges[i][personWhoPaid] =
edges[i][personWhoPaid] - edges[personWhoPaid][i];
            edges[personWhoPaid][i] = 0;
        }
        else if(edges[i][personWhoPaid] ==
edges[personWhoPaid][i])
        {
            edges[i][personWhoPaid] = 0;
            edges[personWhoPaid][i] = 0;
        }
        else
        {
            edges[personWhoPaid][i] =
edges[personWhoPaid][i] - edges[i][personWhoPaid];
            edges[i][personWhoPaid] = 0;
        }
    }
}
else if(option == 2)
{
    int i;
    cout << "Enter number of peoples involved in expense : ";
    cin >> i;

    if(i>n-1 || i<1)
    {
        cout << "Entered number of peoples are not possible."
<< endl;
        return;
    }

    for(int j=1; j<i; j++)

```

```

    {
        string includedName;
        cout << "Enter " << j << "th person : ";
        cin >> includedName;

        if(userMap.count(includedName) == 0)
        {
            cout << "Person " << includedName << " doesn't
exist in group " << group << " !" << endl;
            return;
        }

        int includedPerson = userMap[includedName];

        if(includedPerson == personWhoPaid)
        {
            j--;
            continue;
        }

        double expensePerPerson = expense/i;
        edges[includedPerson][personWhoPaid] =
edges[includedPerson][personWhoPaid] + expensePerPerson;

        if(edges[includedPerson][personWhoPaid] >
edges[personWhoPaid][includedPerson])
        {
            edges[includedPerson][personWhoPaid] =
edges[includedPerson][personWhoPaid] -
edges[personWhoPaid][includedPerson];
            edges[personWhoPaid][includedPerson] = 0;
        }
    }

```

```

        else if(edges[includedPerson][personWhoPaid] ==
edges[personWhoPaid][includedPerson])
        {
            edges[includedPerson][personWhoPaid] = 0;
            edges[personWhoPaid][includedPerson] = 0;
        }
        else
        {
            edges[personWhoPaid][includedPerson] =
edges[personWhoPaid][includedPerson] -
edges[includedPerson][personWhoPaid];
            edges[includedPerson][personWhoPaid] = 0;
        }
    }
}
else
{
    cout << "Option doesn't exist!" << endl;
    return;
}

cout << "\nEnter the Expenses to group " << group << " : ";
cin >> expense;
}
}

void currentPosition(double** edges, int n, unordered_map<int,
string>& ourMap)
{
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
        {

```

```

        if(edges[i][j] != 0)
        {
            cout << ourMap[j] << " wants " << edges[i][j] << "
rupees from " << ourMap[i] << endl;
        }
    }
}

int main()
{
    string group;
    cout << "Enter the group name : ";
    cin >> group;

    int n;
    cout << "Enter number of people in group " << group << " : ";
    cin >> n;

    if(n < 1)
    {
        cout << "Invalid person in group " << group << endl;
        return 0;
    }

    unordered_map<int, string> ourMap;
    unordered_map<string, int> userMap;

    cout << "Enter the names of people in group " << group << " :-" <<
endl;
    for(int i=0; i<n; i++)
    {
        string name;

```

```

        cin >> name;

        ourMap[i] = name;
        userMap[name] = i;
    }

    double** edges = new double*[n];
    for(int i=0; i<n; i++)
    {
        edges[i] = new double[n];
        for(int j=0; j<n; j++)
        {
            edges[i][j] = 0;
        }
    }

    cout << "\n***** MAIN MENU *****" << endl <<
endl;
    cout << "Enter  1 : For add expense to group " << group << endl;
    cout << "Enter  2 : If anyone wants to repay his/her debt." << endl;
    cout << "Enter  3 : For check current position of debt of group " <<
group << endl;
    cout << "Enter -1 : For exit." << endl;

    int i;
    cout << "\nEnter function that you want to perform : ";
    cin >> i;

    while(i != -1)
    {
        switch(i)
        {
            case 1 : addExpense(edges, n, userMap, group);

```

```

        break;

        case 2 : repayDebt(edges, n, userMap, group);
        break;

        case 3 : currentPosition(edges, n, ourMap);
        break;

        default : cout << "Wrong Input!" << endl;
    }

    cout << "*****" << endl <<
endl;
    cout << "\n***** MAIN MENU *****" << endl
<< endl;
    cout << "Enter  1 : For add expense to group " << group <<
endl;
    cout << "Enter  2 : If anyone wants to repay his/her debt." <<
endl;
    cout << "Enter  3 : For check current position of debt of group
" << group << endl;
    cout << "Enter -1 : For exit." << endl;
    cout << "\nEnter function that you want to perform : ";
    cin >> i;
}
cout << "You are Exit." << endl;

for(int i=0; i<n; i++)
{
    delete [] edges[i];
}
delete [] edges;

```

```
    return 0;  
}
```

**GET OUR FULL CODE OF INCLUDING CODE FOR APP IN :-**

<https://drive.google.com/drive/folders/1bvuDsm1QJkZ4ERSoxibcL7UuDUnsawI5?usp=sharing>