Nanuda: A Group Expenses Manager

1st Abia Herlianto
School of Computer Science
BINUS University
Team MONEY
Jakarta, Indonesia
abiaph@live.com

2nd Hugo Ravé
General Engineering
ESILV PARIS-LA DEFENSE
Team MONEY
Paris, France
hugorave07@gmail.com

3rd Nadya Hartanto
School of Information Systems
BINUS University
Team MONEY
Jakarta, Indonesia
ndhartanto1@gmail.com

***Abstract*—Users create a "group" composed of either one person (themselves) or several people. Users then invite these people to the group. Users add expenses and split the costs with either all, some, or no members of the group. Users can add descriptions for each expense. Users can also customise how the costs are split. Users can see a summary of how much they owe to others and how much others owe to others. The summary of how much users owe changes as users spend and pay.**

***Index Terms*—expenses, finance, management**

TABLE I
ROLE ASSIGNMENTS

Role	Name	Task Description
User/Customer	Nadya Hartanto	Tests the application for bugs and issues. Provides feedback to the Development Manager. Also provides feedback on UX and UI quality.
Software Developer	Abia Putrama Herlianto	Implements software based on design. Fixes bugs and issues.
Development Manager	Hugo Ravé	Designs software based on requirements and feedback from User/Customer. Organises, schedules, and manages development.

I. Introduction

When there are shared expenses in a group, it can be complicated to split the costs and gather the money that each person should pay. The usual solution is to have one or several individuals cover the cost of the entire group, and from there the remaining group members would pay their share to those who covered the cost.

The problem that arises from this is keeping track of these expenses. This is especially true for groups that meet regularly; for example, a group of friends going on a trip. Keeping track of who covered what for how much and who owes whom how much can quickly get out of hand. The solution that we propose is a simple application where these expenses can be organised, viewed, and accessed by all members of a certain group.

The name Nanuda comes from Korean 나누다, meaning 'to divide (up), to split (up)'. The name was chosen because our own experiences in splitting expenses in Korea was the reason we thought up of the idea.

There are several existing applications that perform a similar function, with different individual features. These include Tricount and Sesterce.

1) **Tricount**

Fig. 1. Tricount

(Data from Google Play Store)
Rating: 4.8 stars
Downloads: 1 million+
Key features: Users can share a simple link to share their expenses. Members of a group can add expenses and see the balance. Expenses can be shared unevenly. Tricount works both offline and online, with logging in being optional. It also has both a mobile app and a website.

2) **Sesterce**

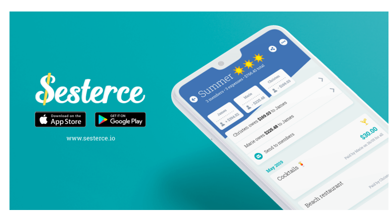

Fig. 2. Sesterce

(Data from Google Play Store)
Rating: 4.7 stars
Downloads: 5,000+
Key features: Similar to Tricount, users can join a single group and add expenses, keeping track of all bills and costs. Sesterce is completely anonymous with no login or email required. In addition, Sesterce allows the addition of custom categories for expenses, the option to view statistics, and exporting all the data to a .csv file.

II. Requirements

1) This application should run on the Android operating system.

2) This application should be able to run offline.
3) **Loading Screen**
   a) The application should load the locally stored groups data.
4) **Groups**
   a) The application should display existing, joined groups.
   b) The user should be able to open the groups.
   c) The user should be able to create new groups and join existing groups.
   d) The user should be able to join a group using a simple link.
   e) The user should be able to edit and delete existing, joined groups.
   f) The user should be able to save changes to and sync with the groups data in the server.
   g) By default, the application should display a sample group.
   h) A group should have one or more participants.
   i) A group should use only one currency out of several options.
5) **Expenses**
   a) The application should display existing expenses in the group.
   b) The user should be able to open the expenses.
   c) The user should be able to create new expenses in the group.
   d) The user should be able to edit and delete existing expenses.
   e) The user should be able to save changes to and sync with the group expenses data in the server.
   f) The user should be able to determine who pays and who owes.
   g) The user should be able to determine how much each participant owes.
6) **Balances**
   a) The application should display how much each participant currently owes or is owed in total.
   b) If a participant owes to another participant, the application should also show how much they owe only to that other participant.

III. Development Environment

A. *Choice of Software Development Platform*

1) **Platform**

Fig. 3. Android

For Nanuda, we have chosen the **Android** mobile operating system as the platform for several reasons:

   a) Nanuda needs to be on a mobile platform for quick access and easy use
   b) Android is the best-selling mobile OS worldwide
   c) Resources for development on Android are widely available and at no cost

2) **Programming Language**

Fig. 4. Java

For the development of Nanuda, we use the **Java** programming language. Java is a class-based, object-oriented programming language and one of the world's most popular and widely used programming languages. We chose to use Java for two reasons:

   a) Java is the language used to develop Android apps through the Android SDK
   b) We are familiar with Java and have used it in the past

3) **Cost Estimation**
For the development of Nanuda, all of the resources used are either free of charge or have both free and paid options. For the resources with free and paid options, we have decided to use the free option. As such, the development of Nanuda requires no cost.

Resource	**Role Description**	**Cost**
Android Studio	Android Development IDE	0
AWS Educate	Backend Server	0
Overleaf	Online, Collaborative LaTeX Writing Tool	0
GitHub	Remote Repository and Version Control System	0
GitHub Desktop	GUI for GitHub	0

4) **Development Environment**
   - **Android Studio Version 4.1**
     Android Studio was chosen because it is the official IDE for the Android operating system.
   - **AWS Educate**
     AWS Educate is Amazon's programme for students to use AWS services at no cost.
   - **Overleaf**
     Overleaf was chosen because of its popularity and the effectiveness of its features, namely simple, in-browser editing, immediate compilation, easy sharing and collaborative editing.

- **GitHub**
  GitHub was chosen for Nanuda's remote repository because of our familiarity with it in comparison to other version control systems.
- **GitHub for Desktop**
  GitHub for Desktop was chosen instead of the regular, CLI-based system to interact with GitHub because of its simplicity and intuitiveness.

B. Software in use

1) **Tricount**

Fig. 5. Tricount

Starts by setting up your account, sign up/login through different ways (Facebook, phone number,etc...). Create your group by choosing a name, purpose of the group (trip, group of friends, couple, family) then choose your currency, add the name of the members (up to 30 participants) and finally share the link of the group to the members and you're all set to start keeping up with your expenses.
Inside a group, you have two main pages: Expenses and Balance. Expenses is where you can see all the expenses of the group as well as the name of the group, who is in it, your total personal expenses and the total expenses of the group. An expense is presented on the screen like this: name of the expense, amount, currency, paid by who and date of the expense. You can click on it to see a picture of the object or receipt if it was added, modify the title of the expense, amount, date, paid by, and for who it was paid. You can also delete the expense. You can sort the expenses by different categories, such as amount paid, date of payment, payer and title.
The second main screen is Balance which shows who owes money and who is owed money. When you click on the owed money to someone, you can mark the debt as paid or you can invite them to pay you through a link sent via messaging platforms.
You also have the main menu screen of the app which allows you to see all the groups you're a part of and where you can add or delete groups.

2) **Sesterce**

The setup is pretty similar to create a group and add members.
There's only 1 main screen in Sesterce. It shows the group name, the number of members, the number of expenses, how much money was spent, who owes money or is owed money by whom. It also shows the list of all expenses sorted by date. The expenses display is pretty similar to Tricount as well, showing the title, the amount of the expense, who it is paid by and for whom. You can also add expenses. You have to chose who paid (one or multiple persons), the amount paid by each person, then select a title, add a date and choose a category for the expense (accommodation, groceries, transportation, you can even add your own category).

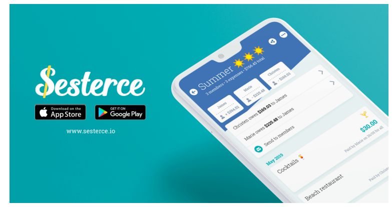

Fig. 6. Sesterce

IV. Specifications

1) **Loading Screen**
   a) The application should load the locally stored groups data.
   b) If it is the first time the application is opened, the application should create new locally stored groups data. By default, a sample group with sample expenses should be made. After it is created, the new locally stored groups data should be uploaded to the server and added to the database.
2) **Groups**
   a) Groups List
      i) The application should display a list of the existing locally stored groups.
      ii) For each group, its name and description are displayed.
      iii) Next to each group, there should be a delete button which if clicked should delete the group from the locally stored groups data.
      iv) The user should only be part of no more than 5 groups.
   b) Enter Group
      i) If the user clicks on a group, the application should move to the group's Expenses screen.
   c) Add Group
      i) If the user clicks the "Add Group" button, the application should display the "Add Group" pop up screen.
      ii) The application should display the "Make New Group" and "Join Group" buttons.
      iii) If the user is already part of 5 groups, the application should display an error informing the user that they have reached the maximum number of groups.
   d) Make New Group

i) If the user clicks the ”Make New Group” button, the application should display the ”New Group” pop up screen.
ii) The user should input the name of the new group, choose a currency to be used by the expenses in the group, and input the names of the participants of the new group.
iii) The user should optionally be able to add a description to the new group.
iv) The name of the new group should not be empty and should be no longer than 20 characters.
v) If there is a description, the description of the new group should be no longer than 50 characters.
vi) The user should be able to choose one of at least three major currencies for the new group.
vii) The user should be able to add participants' names by inputting the name and clicking the ”Add” button.
viii) The new group should have no more than 20 participants.
ix) If the user clicks the ”Create New Group” button, the application creates the new group and display a link to share to other users to join the group.
x) If the requirements for the new group have not been met, the ”Create New Group” button should not be interactable.
xi) If the user clicks the ”Back” button, the application should move back to the groups list screen.

e) Join Group
i) If the user clicks the ”Join Group” button, the application should display the ”Join Group” pop up screen.
ii) The user should input the link of the group they wish to join.
iii) If the user clicks the ”Join” button, the application sends a request to the server to find the group and download its information.
iv) If the link has not been input or the link is invalid, the ”Join” button should not be interactable.
v) If the user clicks the ”Back” button, the application should move back to the groups list screen.

f) Sync Groups
i) If the user clicks the ”Sync” button, the application should download the latest groups data from and upload changes to the server.

3) **Expenses**

a) Back Button
i) If the user clicks the ”Back” button, the application should move back to the groups list screen.

b) Balances Button
i) If the user clicks the ”Balances” button, the application should move to the balances screen.

c) Edit Group
i) If the user clicks the ”Edit Group” button, the application should display the ”Edit Group” pop up screen.
ii) The application should display the name, currency used by, and names of the participants of the group.
iii) The user should be able to change the name, the description, the currency used by the expenses in the group, and the names of the participants of the group.
iv) The name of the group should not be empty and should be no longer than 20 characters.
v) If there is a description, the description of the group should be no longer than 50 characters.
vi) The user should be able to choose one of at least three major currencies for the new group.
vii) The user should be able to add participants' names by inputting the name and clicking the ”Add” button.
viii) The group should have no more than 20 participants.
ix) If the user clicks the ”Save Changes” button, the application updates the group and moves back to the expenses list screen.
x) If the requirements for the group have not been met, the ”Save Changes” button should not be interactable.
xi) If the user clicks the ”Back” button, the application should move back to the expenses list screen.

d) Expenses List
i) The application should display a list of the existing locally stored expenses of the group.
ii) For each expense, its name, amount, payer, and date should be displayed.
iii) Next to each expense, there should be a delete button which if clicked should delete the expense from the locally stored data.
iv) There should be no limit to the number of expenses in a group.

e) Enter Expense
i) If the user clicks on an expense, the application should display the details of the expense.
ii) The application should display the name, amount, date, and payer of the expense. It should also display the participants who were paid for by this expense and how much each participant owes. It should also display the ”Edit Expense” button.

iii) If the user clicks the ”Back” button, the application should move back to the expenses list screen.

f) Edit Expense

i) If the user clicks on the ”Edit Expense” button, the application should display the ”Edit Expense” pop up screen.

ii) The application should display the name, amount, and date of the expense and choose the participant who pays and participants who are paid for.

iii) The user should be able to change the name, amount, date of the expense, the participant who pays, and the participants who are paid for.

iv) The name of the expense should not be empty and should be no longer than 20 characters.

v) The amount of the expense should not be empty.

vi) The payer participant should not be empty and the user should choose a participant from the list of participants.

vii) The application should display all the participants in the group under the ”Paid For” label.

viii) Next to each participant, there should be a checkbox so the user can choose whether the participant is paid for or not.

ix) There should be a checkbox next to the ”Paid For” label to choose all participants in the group.

x) Next to each participant whose checkbox is ticked, there should be the amount that will be owed by the participant. By default, the total amount expended should be divided equally between all paid for participants.

xi) The user should optionally be able to change how much each participant will owe in the expense manually.

xii) If the user edits how much each participant will owe in the expense manually, the total should be equal to the amount expended by the payer participant.

xiii) If the user clicks the ”Save Changes” button, the application updates the expense and moves back to the expenses list screen.

xiv) If the requirements for the new expense have not been met, the ”Save Changes” button should not be interactable.

xv) If the user clicks the ”Back” button, the application should move back to the expenses list screen.

g) Add Expense

i) If the user clicks the ”Add Expense” button, the application should display the ”New Expense” pop up screen.

ii) The user should input the name, amount, and date of the expense and choose the participant who pays and the participants who are paid for.

iii) The name of the expense should not be empty and should be no longer than 20 characters.

iv) The amount of the expense should not be empty.

v) By default, the date of the expense should be the current date.

vi) The payer participant should not be empty and the user should choose a participant from the list of participants.

vii) The application should display all the participants in the group under the ”Paid For” label.

viii) Next to each participant, there should be a checkbox so the user can choose whether the participant is paid for or not.

ix) There should be a checkbox next to the ”Paid For” label to choose all participants in the group.

x) Next to each participant whose checkbox is ticked, there should be the amount that will be owed by the participant. By default, the total amount expended should be divided equally between all paid for participants.

xi) The user should optionally be able to input how much each participant will owe in the expense manually.

xii) If the user edits how much each participant will owe in the expense manually, the total should be equal to the amount expended by the payer participant.

xiii) If the user clicks the ”Add Expense” button, the application creates the new expense and moves back to the expenses list screen.

xiv) If the requirements for the new expense have not been met, the ”Add Expense” button should not be interactable.

xv) If the user clicks the ”Back” button, the application should move back to the expenses list screen.

4) **Balances**

a) Back Button

i) If the user clicks the ”Back” button, the application should move back to the groups list screen.

b) Expenses Button

i) If the user clicks the ”Expenses” button, the application should move to the expenses screen.

c) Summary

i) The application should display the name of each participant in the group and how much they owe or are owed to.

ii) If in total the participant owes more than they are owed, then the total amount they owe should be displayed with a red background.

iii) If in total the participant is owed more than they owe, then the total amount they are owed should be displayed with a green background.

d) Balancing Details

   i) The application should display how much each participant owes another participant in separate balances. If A owes B and C, and B owes C, then the application should display how much A owes B, how much A owes C, and how much B owes C in separate balances.