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 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.

II. REQUIREMENTS

1) This application should run on the Android operating system.
2) **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.
3) **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.
4) **Balances**
   a) The application should display how much each participant currently owes or is owed in total.
   b) If participants owe to each other, the application should show how to settle the owed amounts.

III. Development Environment

A. Choice of Software Development Platform

1) **Platform**

Fig. 1. 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. 2. 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.

TABLE II
Cost Estimation

Resource	Role Description	Cost
Android Studio	Android Development IDE	0
Back4App	Backend Server	0
Overleaf	Online, Collaborative LaTeX Writing Tool	0
GitHub	Remote Repository and Version Control System	0
GitHub Desktop	GUI for GitHub	0
MockFlow	UI Planning Platform	0

4) **Development Environment**

- **Android Studio Version 4.1**

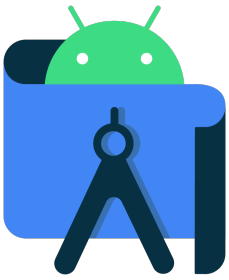

Fig. 3. Android Studio

Android Studio was chosen because it is the official IDE for the Android operating system.

- **Back4App**

Fig. 4. Back4App

Back4App is a low-code backend based on the open-source Parse Platform. Back4App was chosen due to its accessibility and ease of use, allowing anyone to make apps faster without managing infrastructure. It can easily be integrated on any mobile or web app through GraphQL, Rest APIs, or the various Parse SDKs.

- **Overleaf**

Fig. 5. 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**

Fig. 6. 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**

Fig. 7. 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.

- **MockFlow**

Fig. 8. MockFlow

MockFlow is a powerful online UI planning platform. MockFlow was chosen due to its rich premade library of components as well as its powerful editor, even in its free package.

B. Software in use

1) **Tricount**

Fig. 9. Tricount

- **Rating**: 4.8 stars (48,000 ratings) on Google Play Store
- **Downloads**: 1 million+ on Google Play Store
- **Features**:
  - Optional log in feature allows all a user's Tricounts to be linked to a user's profile
  - Share simple link so other users can join groups and see expenses
  - Anyone in a group can add their own expenses
  - Splitting expenses unevenly
  - Works offline
  - Works on both the web and mobile (Android, iOS, and Windows Phone)
  - Optional push notifications when someone edits or adds expenses
  - Sort expenses by categories
  - Several other paid features such as statistics
- **Differences**:

– Nanuda does not have log in feature to be as lightweight as possible
– Only mobile app
– No push notifications
– No expense categories
– Does not have some of Tricount's paid features such as statistics

2) **Sesterce**

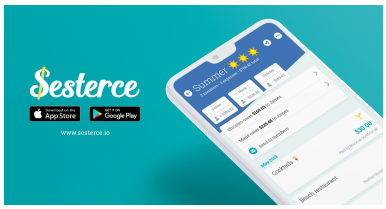

Fig. 10. Sesterce

- **Rating**: 4.7 stars (106 reviews) on Google Play Store
- **Downloads**: 5,000+ on Google Play Store
- **Features**:
  – Splitting expenses unevenly
  – Sort expenses by categories
  – View statistics by category and group member
  – Converting foreign currency
  – Export all data
- **Differences**:
  – Nanuda does not have expense categories
  – No statistics
  – No conversion between currencies
  – Cannot export data

C. Task distribution

TABLE III
TASK DISTRIBUTION

Name	Task
Abia Putrama Herlianto	Balances Frontend, Backend, Documentation
Hugo Ravé	Groups Frontend, Documentation
Nadya Hartanto	Expenses Frontend, Documentation

IV. SPECIFICATIONS

1) **Groups**

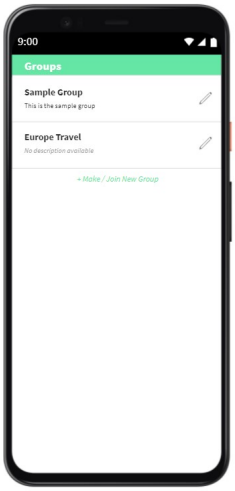

Fig. 11. Groups List Screen

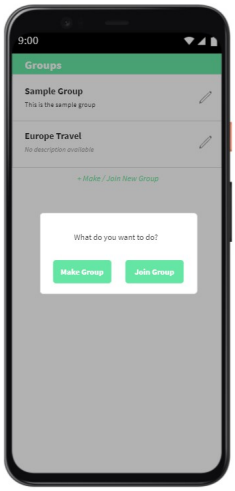

Fig. 12. Make New Group Screen

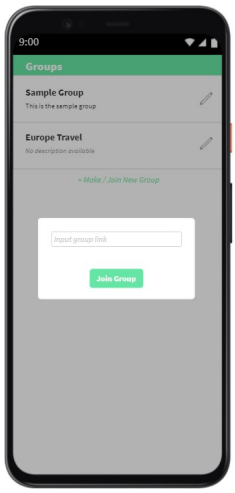

Fig. 13. Join Group Screen

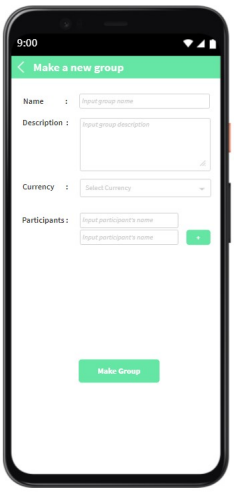

Fig. 14. Make Group Screen

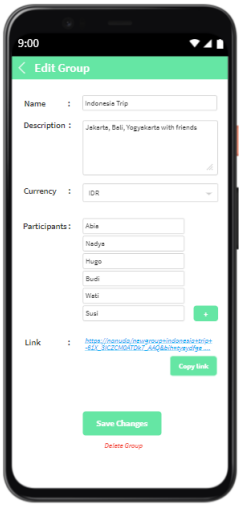

Fig. 15. Edit Group Screen

a) Groups List
   i) The application should load the locally stored group ID data.
   ii) If it is the first time the application is opened, the application should create new locally stored group ID data. By default, a sample group with sample expenses should be made. After it is created, the new sample group should be uploaded to the server and added to the database.
   iii) The application should display a list of the existing groups.
   iv) For each group, its name and description are displayed.
   v) 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) 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.

g) Sync Groups

i) If the user clicks the ”Sync” button, the application should download the latest groups data from the server.

h) Delete Groups

i) If the user long-presses any group, the user should be able to select groups to be deleted.
ii) If the user selects groups to be deleted, there should be a ”Delete Groups” button to delete the groups.

2) **Expenses**

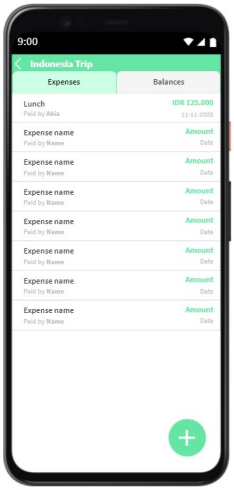

Fig. 16. Expenses Main Screen

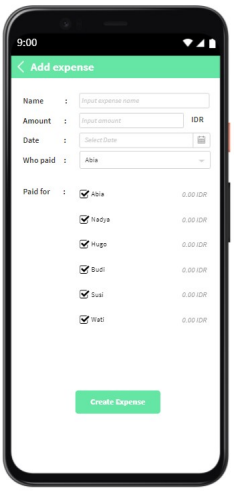

Fig. 17. Add Expense Screen

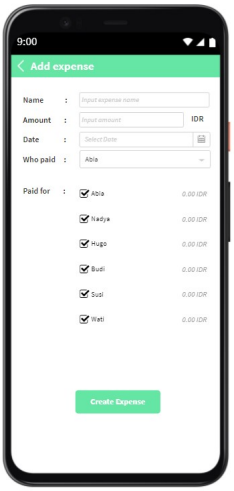

Fig. 18. Edit Expense Screen

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) Expenses List
   i) The application should display a list of the existing expenses of the group.
   ii) For each expense, its name, amount, payer, and date should be displayed.
   iii) There should be no limit to the number of expenses in a group.
d) 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.
e) 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.
f) 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.

g) Sync Expenses

i) If the user clicks the ”Sync” button, the application should download the latest group expenses data from the server.

3) **Balances**

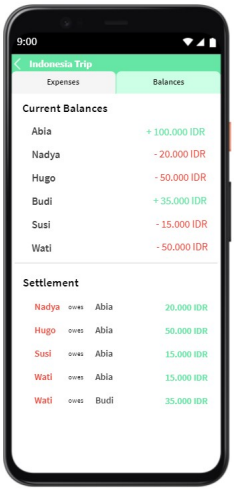

Fig. 19. Balances Screen

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.

V. Architecture Design and Implementation

A. Overall architecture

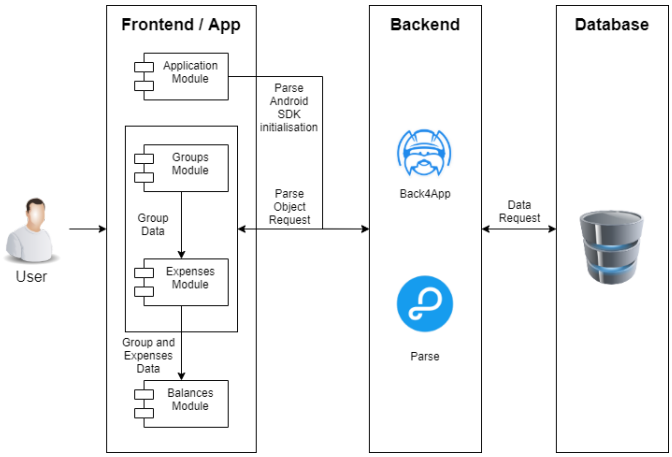

Fig. 20. Overall Architecture

The Nanuda Android application is divided into four separate modules.

The first part is the ”Application” module which is responsible for setting up and starting the application. The ”Application” module does this by initialising the application, splash screen, and everything required to connect to the backend. Nanuda uses the low-code back4app open source Backend as a Service provider for its backend, which means Nanuda does not have a separate backend module. Instead, the ”Application” module is responsible for connecting and allowing itself as well as other modules to connect to the backend.

The second part is the ”Groups” module which is responsible for displaying, adding, joining, editing, and deleting groups. It interacts with the backend to get, add, edit, and remove group data.

The third part is the ”Expenses” module which is responsible for displaying, adding, editing, and deleting expenses. It interacts with the backend to get, add, edit, and remove expense data.

The fourth part is the ”Balances” module which is responsible for calculating and displaying the summaries of how much each user owes or is owed in total and how users can settle them.

These four modules interact with each other as well as the backend in order for the application to function.

B. Directory organization

The organisation of Nanuda’s directories follows the organisation of an Android Studio project. Generally, the organisation of the Java packages in the Android Studio project closely follows the overall architecture which splits the application into four modules. In addition, there is a separate directory for files related to the documentation of the project.

TABLE IV
DIRECTORY ORGANIZATION

Directory	File Name	Module Name
\app\src\main \java\com \teammoney \nanuda	Nanuda.java SplashScreenActivity.java	Application
\app\src\main \java\com \teammoney \nanuda\objects	Expense.java Group.java	Application
\app\src\main \java\com \teammoney \nanuda\expense	ExpensesListActivity.java ExpenseAdapter.java MakeExpenseActivity.java EditExpenseActivity.java	Expenses
\app\src\main \java\com \teammoney \nanuda\group	GroupsListActivity.java MakeGroupActivity.java EditGroupActivity.java	Groups
\app\src\main \java\com \teammoney \nanuda\balances	BalancesActivity.java BalancesAdapter.java	Balances
\docs	documentation.pdf documentation.tex	Documen -tation
\app\src\main \res\drawable	ic_baseline_arrow_back_ios_24.xml ic_launcher_background.xml splash_screen_background.xml	Application Groups Expenses Balances
\app\src\main \res\layout	activity_edit_expense.xml activity_edit_group.xml activity_expenses_lists.xml activity_groups_list.xml activity_make_expense.xml activity_make_group.xml activity_splash_screen.xml expenses_list_item.xml group_toolbar.xml join_popup.xml make_new_group_toolbar.xml	Application Groups Expenses Balances
\app\src\main \res\menu	join_group_menu.xml	Groups
\app\src\main \res\values	colors.xml dimens.xml ids.xml strings.xml themes.xml	Application Groups Expenses Balances

C. Module 1: Application

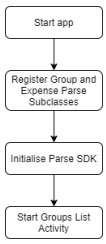

Fig. 21. Application Module Flowchart

- Purpose
  The Application module is responsible for initialising the app and its connection with the backend as well as defining information required by the app to function properly.
- Functionality
  It starts the app, connects it with the backend, and defines the Parse Objects to be used by the Android Parse SDK by the application.
- Location of Source Code
  - nanuda\app\src\main\java\com\teammoney\nanuda
  - nanuda\app\src\main\java\com\teammoney\nanuda\objects
- Class Components
  - Nanuda.java
    Registers the Parse Object subclasses ”Group” and ”Expense” as required by the SDK and initialises the Parse SDK with the app’s application ID, client key, and connects it to the backend at back4app.
  - SplashScreenActivity.java
    Sends an intent from the Splash Screen to the Groups List to start the Groups List Activity.
  - Group.java
    Defines the Group Parse Object subclass as the local Android implementation of the Group table in the backend.
  - Expense.java
    Defines the Expense Parse Object subclass as the local Android implementation of the Expense table in the backend.
- Where It’s Taken From
  We developed the module ourselves using resources from Back4App and Parse Platform’s documentation at https://www.back4app.com/docs and https://docs.parseplatform.org/android/guide respectively.
- How/Why We Use It
  The Groups module was made because it is one of the three main features of Nanuda. To make it into its own module with its own functionalities and required data is intuitive and makes the development simpler.

D. Module 2: Groups

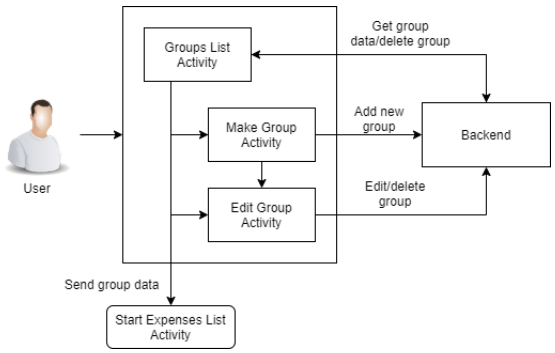

Fig. 22. Groups Module

- Purpose
  The Groups module is responsible for displaying, making, joining, editing, and deleting groups.

- Functionality
  It gets a list of groups the user has joined from the locally-stored file groups.txt and sends a query to the backend to get the groups' data. It displays all these groups. It also allows the user to make a new group and join, edit, and delete existing groups.
- Location of Source Code
  nanuda\app\src\main\java\com\teammoney\nanuda\group
- Class Components
  - GroupsListActivity.java
    Reads the group IDs of the groups the user has joined from the locally-stored groups.txt file and sends a query to the backend to get the groups' data. Using the group data, it creates a Recycler View with each group as a list item. For each list item, it sets up an OnClickListener so that if a user clicks a group, it will send an intent to the Expenses List to start the Expenses List activity. It also sets up the the "Make/Join Group" button, the "Make/Join Group" popup, and the "Join Group" popup. Also sets up the functionality for group list items so that when they are long-pressed, they show a delete option.
  - MakeGroupActivity.java
    Sets up the make group form and sends a request to the backend to create a group using the provided information.
  - EditGroupActivity.java
    Sets up the edit group form with the group's existing data and sends a request to the backend to edit the group using the provided information.
- Where It's Taken From
  We developed the module ourselves using resources from Back4App and Parse Platform's documentation at https://www.back4app.com/docs and https://docs.parseplatform.org/android/guide respectively.
- How/Why We Use It
  The Expenses module was made because it is one of the three main features of Nanuda. To make it into its own module with its own functionalities and required data is intuitive and makes the development simpler.

E. Module 3: Expenses

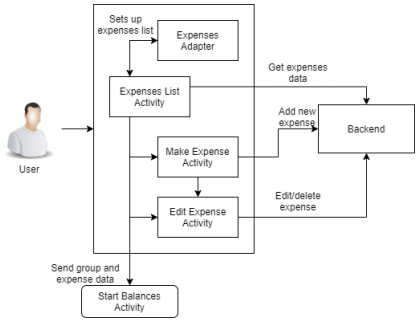

Fig. 23. Expenses Module

- Purpose
  The Expenses module is responsible for displaying, making, editing, and deleting expenses.
- Functionality
  It receives the Group data, queries for all the expenses associated with that group, and displays them. It also allows the user to add new expenses and edit and delete existing ones.
- Location of Source Code
  nanuda\app\src\main\java\com\teammoney\nanuda\expense
- Class Components
  - ExpensesListActivity.java
    Sets the "Expenses" tab as the selected tab, adds an OnTabSelectedListener to the "Balances" tab so that when the user selects it it will send an intent to the Balances and start the Balances Activity, gets the group data and queries the backend for all expenses associated with that group and passes that data to the Expenses Adapter to make it into a Recycler View.
  - ExpensesAdapter.java
    Adapts the expenses list provided by ExpensesListActivity.java into Recycler View form.
  - MakeExpenseActivity.java
    Sets up the make expense form and sends a request to the backend to create an expense using the provided information.
  - EditExpenseActivity.java
    Sets up the edit expenses form with the expense's existing data and sends a request to the backend to edit the expense using the provided information.

F. Module 4: Balances

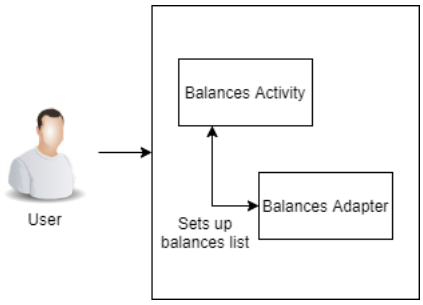

Fig. 24. Balances Module

- Purpose
  The Balances module is responsible for calculating and displaying the summary of how much each user owes or is owed and how to settle them.
- Functionality
  It receives the Group and Expenses data, counts the summary for each user, calculates how participants in debt should pay the other participants that are indebted to.
- Location of Source Code
  nanuda\app\src\main\java\com\teammoney\nanuda\balances

- Class Components
  - BalancesActivity.java
    Sets the ”Balances” tab as the selected tab, adds an OnTabSelectedListener to the ”Expenses” tab so that when the user selects it it will send an intent to the Expenses List and start the Expenses List Activity, calculates the total of how much each user owes or is owed, how to settle them, and passes that data to the Recycler View adapter. To calculate the total of how much each user owes or is owed, the algorithm goes through each Expense and adds it to or subtracts it from the payer's total. To calculate how to settle it, the algorithm sorts the users who owe and the users who are owed to in descending order separately. Then starting from the user who is owed the most, the algorithm goes through the users who owe and has them pay the amount owed. It continues this until the entire amount owed is accounted for.
  - BalancesAdapter.java
    Adapts the summary list and details list provided by BalancesActivity.java into Recycler View form. In order to optimise the code, both lists as well as their headers are part of a single long RecyclerView. First, the header of the summary list is entered into the Recycler View, followed by the summary list items, the header of the details list, and finally the details list items.
- Where It's Taken From
  We developed the module ourselves using resources from Back4App and Parse Platform's documentation at https://www.back4app.com/docs and https://docs.parseplatform.org/android/guide respectively.
- How/Why We Use It
  The Balances module was made because it is one of the three main features of Nanuda. To make it into its own module with its own functionalities and required data is intuitive and makes the development simpler.