Requirement & Analysis Document

1. Introduction

The project aims to create a personal finance application that serves the purpose of handling debt between participants in a certain group. Keeping track of shared expenses between friends has never been easier.

The application is primarily aimed toward people who appreciate the convenience of smartphones and who want to utilize this to further organize their lives.

Debtify is not aimed for business or professional use, but rather day-to-day casual use. It creates value for users by simplifying the menial and in some instances uncomfortable tasks of managing debts. By using the application, the experience of sharing expenses becomes more organized and enjoyable with an intuitive interface and pleasing design.

1.1 General characteristics of the application

- The application is an Android app written in Java.
- Users can register a new account with a name, phone number, and password.
- Users can add other users to their contacts.
- Users can create a group and add users from their contacts to the group.
- Users in a group can add new debts/payments to another user in the same group.
- Users can see a sorted log of all transactions in a group.

1.2 Scope of application

- No actual financial value will be transferred through the application. It will resemble a simplified ledger for personal debt between participants.
- The application will have access to a remote database where the application retrieves and stores data.

1.3 Objectives and success criteria of the project

1. Users should be able to create an account.
    a. With a name, a phone number, and a password.
2. Users should be able to create a group and add other people to the group.
    a. Group has a name.
    b. Users can be added to the group.
3. Users should be able to add debt to others.
    a. Either individually or collectively as a group.
4. It should be possible to see the most recent transactions.

1.4 Definitions, acronyms, and abbreviations

- App
    - Abbreviation of application.
- View
    - A displaying page that the user can see.
- Endless Decimal:
    - A number which is not representable in decimals. As such the number will never be completely accurate. The most common example of this might be 10 divided by 3, resulting in 3,[an endless amount of threes].
- Hamburger-button
    - Is generally a button in the top-corner that unintentionally resembles a hamburger. That is, three horizontal lines stacked on each other.
- Notification
    - A message that pops up on a mobile device.
- Special Sign
    - A symbol which is not in the range of [A-Z], [a-z] or [0-9].
- JUnit
    - Java's unit testing framework.
- Master-branch
    - The repository's default and final branch. This is where the latest version of the application is found.

2. Requirements

2.1 User Stories

Initial user stories that the project was based on.

High Priority	Medium Priority	Low Priority
US01: Show debts for specific groups.	US07: Debt simplifier in Group.	US09: Activity feed.
US02: Create an account.	US12: My profile page.	US13: Graph functionality for debt in each group
US03: Log in.	US10: Add personal contacts.	US14: Graph functionality for debt for the logged-in account.
US04: Create groups.	US11: Access contacts from storage on the device	US15: Debt simplifier between individuals across groups
US05: Add user(s) to groups.		US16: Notification of actions involving debt
US06: Add debt to user(s) in a group.		

US08: Settle debt between users in a group.		
US19: Leave group.		

Figure 2.0: Completed User Stories are marked in green

High Priority

US01: Show debts for specific groups.

Implemented?

- **Yes**

Description

- As a user, I want to see all the added debts in a certain group so I can understand my debt-situation.

Acceptance criteria

- Functional
  i. There exists a way to reach the View for the log of debts.
  ii. There exists a View for the log of debts.
  iii. The debts must be logged somewhere to be able to display them.
  iv. The debts are sorted by date and viewable.
  v. There exists a way to exit the View.
- Non-functional
  i. The debt will be shown on a card.
  ii. The maximum amount of debts will be 4.
  iii. If the list of debts exceeds 4, the user will be able to scroll down to see more debts.

US02: Create an account

Implemented?

- **Yes**

Description

- As a user, I want to be able to create an account so I can use the application and save my personal data.

Acceptance criteria

- Functional
  i. If the user is not logged in, the user is supplied with a button that leads the user to create an account.
  ii. The user is prompted to enter Phone number, Name, and Password on account creation.
  iii. After successfully registering, the data is automatically stored.
- Non-functional
  i. The user's credentials need to be stored in a database if the registration is successful and the user is connected to the internet.
  ii. If the user is not connected to the internet, the user shan't be able to register.

US03: Log in

Implemented?

- **Yes**

Description

- As a registered user, I want to log in so I can access my account and my saved data.

Acceptance criteria

- Functional
    i. The user will be introduced to the login page if the user is not logged in.
    ii. On the login page, a user is prompted to enter the Phone number and password and can subsequently log in.
    iii. There exists a way to log out of the account.
- Non-functional
    i. If the user is not connected to the internet, the user shan't be able to log in.

US04: Create groups.

Implemented?

- **Yes**

Description

- As a user who owes/lends money to other user(s), I want to create a group of users so that I can keep track of my/their debt.

Acceptance criteria

- Functional
    i. There exists a group creation View.
    ii. There exists a way to navigate to the group creation View.
    iii. In the group creation View, the user can specify the name of the group.
    iv. In the group creation View, the user can invite other users to the group.
    v. Once the group is created, the user should be able to see it somewhere.
- Non-functional
    i. If the user is in several groups, the user should be able to scroll to see them all.

US05: Add user(s) to groups .

Implemented?

- **Yes**

Description

- As a user who just met my significant other, I want to add them to a group so I can put them in debt.

Acceptance criteria

- Functional
    i. There exists a way to add users to a group.
    ii. Ability to add other users to the group.

iii. Multiple users should be able to be added at once.

- Non-functional
  i. The added user does not have to accept anything and joins the group immediately.

US06: Add debt to user(s) in a group.

Implemented?

- **Yes**

Description

- As a user, I want to put one or more users - in a group - in debt so that I can track it in the future.

Acceptance criteria

- Functional
  i. There exists a View where the user can create debt.
  ii. There is a way in the group view to navigate to the debt creation View.
  iii. The user can put multiple users in debt at once.
  iv. The user can specify the total amount of debt.
  v. The user can specify the reasoning behind the debt.
  vi. The user can choose to split the debt equally to the selected users.
  vii. The user can send the debt to the selected users.
- Non-functional
  i. The debt creation button should pop out with a distinct color.
  ii. The debt should be added online and therefore backed up before it shows up as added in the group.

US08: Settle debt between users in a group.

Implemented?

- **Yes**

Description

- As a user, I want to settle a pending debt between one or more users in a group so that I am no longer indebted.

Acceptance criteria

- Functional
  i. There exists a View where the user can select and settle a pending debt.
  ii. The user can specify how much of the debt to settle.
  iii. The user cannot settle more than the existing debt amount.
  iv. The debt is updated after a settlement.
- Non-functional
  i. The pay button should pop out with a distinct color.
  ii. The payment should be added online and therefore backed up before it shows up as added in the group.

US19: Leave group.

Implemented?
- **Yes**

Description
- As a user, I want to be able to leave a group that I no longer want to be a part of.

Acceptance criteria
- Functional
    i. There exists a button where the user can use to leave the specified group.
- Non-functional
    i. All debts and lendings associated with the user will cease to exist when the user has left the group.

Medium Priority

US07: Debt simplifier in Group.

Implemented?
- No

Description
- As a user who is bad at math, I want the app to simplify the overall debt so that I can see debts in a comprehensible way.

Acceptance criteria
- Functional
    i. There exists a button that displays the simplified view of debts in a group when pressed.
- Non-functional
    i. The debts shall be simplified in the following ways.
        1. It only simplifies debts related to the logged-in user.
        2. If user A owes money to user B and user B owes money to user C, then user A owes money to user C, if and only if the amount of money user A owed to user B is less or equal to the amount user B is owed to user C. Else user A just owes to user B.
    ii. The simplifier is reliable to the degree that it's reversible with an error of 0.001 units when working with endless decimals.

US12: My profile page

Implemented?
- No

Description
- As a user, I want to be able to see and edit my profile to keep it updated.

Acceptance criteria
- Functional
    i. Can I see my name on my profile?
    ii. Can I see my picture on my profile?

iii. Can I see a biography in my profile?
iv. Can I edit the information on my profile page?
1. Name
2. Image
3. Biography

- Non-functional
  i. Monitorability
  1. Can I always see my name on my profile page?
  2. Can I always see my image on my profile page?
  3. Can I always see my biography on my profile page?
  4. Can I easily change to edit-mode on my profile page?
  ii. Security
  1. If text fields are left empty, will I not be able to save the edited information then?
  2. Are special signs allowed?

US10: Add personal contacts

Implemented?

- **Yes**

Description

- As a user with an account, I want to be able to add personal contacts so that I can save my acquaintances' information and quickly access them in groups.

Acceptance criteria

- Functional
  i. There exists a contact view that can be accessed.
  ii. On the contacts-page, all added contacts are listed.
  iii. On the contacts page, there exists a way to add and remove a contact.
- Non-functional
  i. The user can never add the same user twice.
  ii. The user can add an infinite amount of contacts.

US11: Access contacts from storage on the device

Implemented?

- No

Description

- As a user, I want to be able to add a new person through the inbuilt contacts (in the phone) so that I don’t have to manually write in their information.

Acceptance criteria

- Functional
  i. Able to see contacts.
  ii. Able to select a contact.
  iii. The app should be able to see if it is an existing contact.
  iv. The app should verify that the added contact has an account on the app.
- Non-functional

i. The view for showing contacts from the phone's contact book will show at most five contacts at a time.

Low Priority

US09: Activity feed.

Implemented?
- No

Description
- As a user, I want to be able to see a feed of the most recent events so that I can keep myself updated.

Acceptance criteria
- Functional
  i. When the user is logged in to the application, it is greeted by an Activity feed.
  ii. The Activity feed displays all events/actions done by the logged-in user.
- Non-functional
  i. The activity feed will show the recent events in reverse-chronological order.

US13: Graph functionality for debt in each group

Implemented?
- No

Description
- As a user, I want to be able to see all the debts in a group displayed in a graph so that I can easily understand it.

Acceptance criteria
- Functional
  i. There is a way to navigate to the group-specific debt-graph.
  ii. There is also a graph for displaying debt distribution by category.
  iii. All the debts in the groups are displayed in a graph.
- Non-functional
  i. The graph is shown as a pie.

US14: Graph functionality for debt for the logged-in account.

Implemented?
- No

Description
- As a user, I want to be able to see all the debts to individuals displayed in a graph so that I can more easily understand it.

Acceptance criteria
- Functional
  i. There exists a way to navigate to the graph.
  ii. The graph displays a history of debts.

- Non-functional
    i. The graph displays the history for up to 12 months.

US15: Debt simplifier between individuals across groups

Implemented?
- No

Description
- As a user, I want my debts to a specific user across all mutual groups to be calculated to what I owe that user in total, so I can pay that user.

Acceptance criteria
- Functional
    i. If a debt is added to a group or a user in a group, from a user, the exact debt to each user is calculated.
    ii. There exists functionality to simplify the debt between users.
- Non-functional
    i. The simplifier is reliable to the degree that it's reversible with an error of 0.001 units when working with endless decimals.

US16: Notification of actions involving debt

Implemented?
- No.

Description
- As a user, I want to be notified of all actions done to the debts that affect me so I won't miss it.

Acceptance criteria
- Functional
    i. If a debt is added to a user or more, a notification will be sent to all users affected by it.
    ii. If a debt is settled between two or more users, a notification will be sent to all users affected by it.
- Non-functional
    i. The notification shall be sent as soon as possible whenever the user is logged in and is connected to the internet.

2.2 Definition of Done

- The code is peer-reviewed.
- Task has met the acceptance criteria.
- The code has passed JUnit tests.
- The application builds and compiles.
- The code is documented (JavaDoc).
- Merged into master-branch.
- Updated UML.

2.3 User Interface

The focus of this section is to describe the navigation among the views and to visualize the GUI through screenshots.

2.3.1 Starting Page

The user is greeted by the login page where the user has the opportunity to register a new account or log in to an existing account.

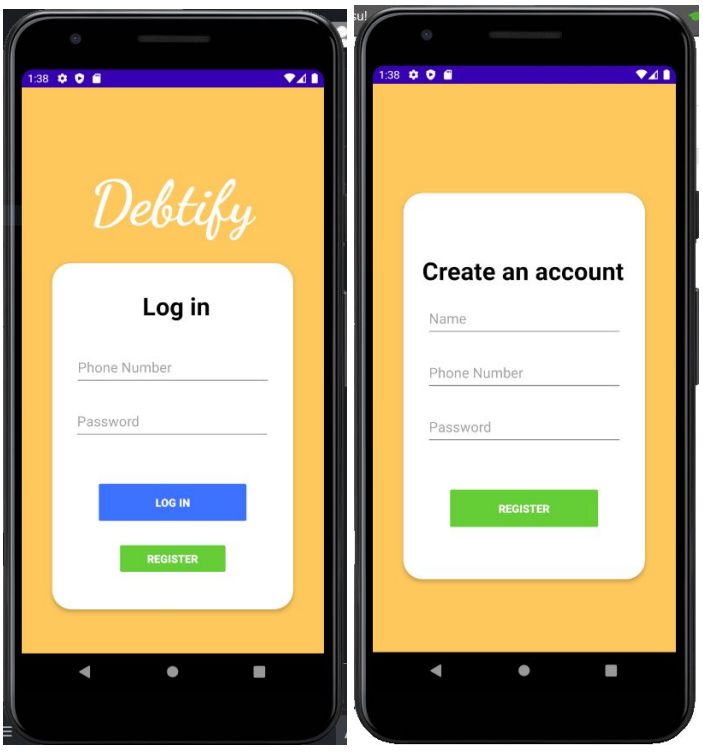


Figure 2.1: LoginView to the left, RegisterPage to the right.

2.3.2 Dashboard

After successfully logging in, the dashboard is displayed which contains minor user info at the top, a view that shows all of the groups - as card views - that are associated with the logged-in user, and the total debt balance for the user among the groups. There's also a floating action button in the bottom right which navigates to the group creation page.

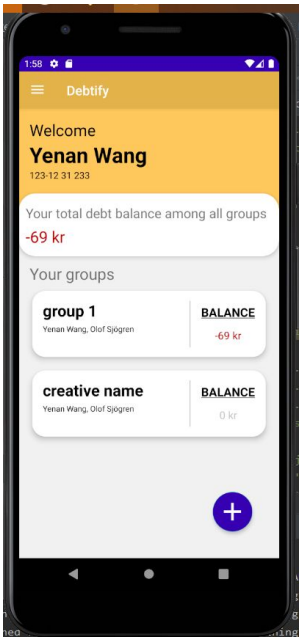


Figure 2.2: Dashboard.

2.3.3 Drawer Layout

A side panel appears by pressing the hamburger-button in the top-left corner of the dashboard, or alternately by dragging from the left edge to the right. The side panel acts as the application's global navigation.

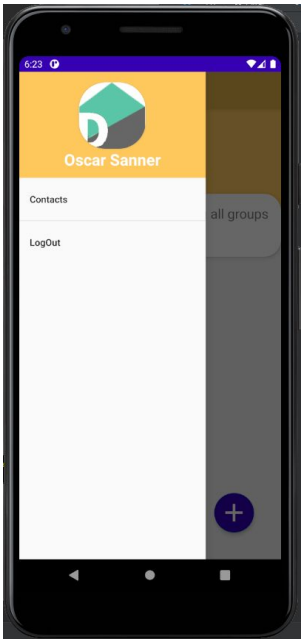


Figure 2.3: Drawerlayout which is accessed through the hamburger-button.

2.3.4 Contacts Page

The contacts page is accessed through the drawer layout which displays the currently logged-in user's contacts. The user can also add or remove contacts.

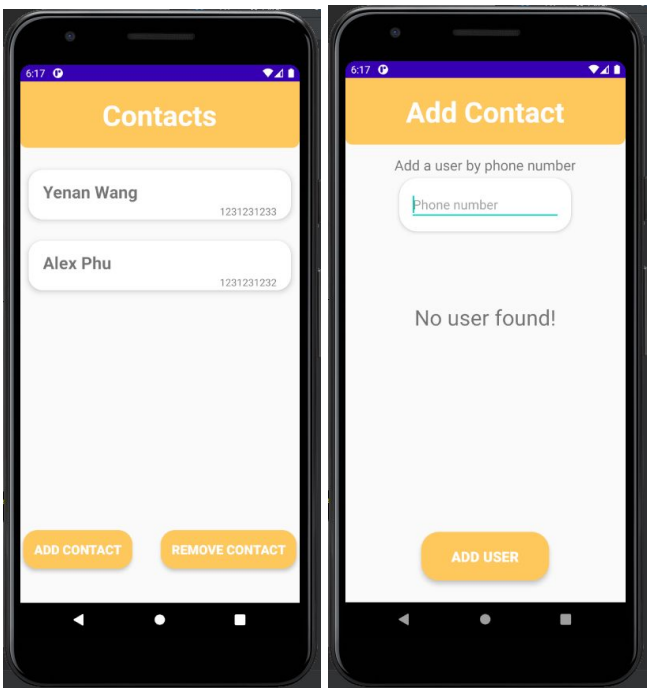


Figure 2.4: Contacts page to the left, page to add a contact to the right.

2.3.5 Create a Group Page

The group creation page is reached by pressing the floating action button. The user has to initially pick the participants of the new group before being able to name the group.

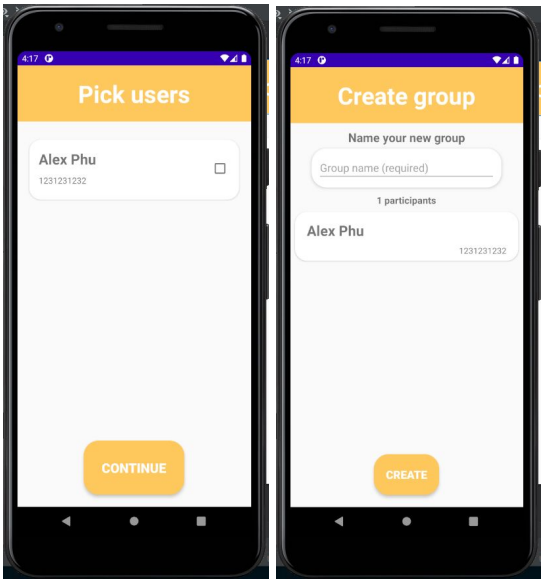


Figure 2.5: Pick user page on the left, page to name on the right.

2.3.6 Detailed Group View

The user enters the detailed view of a group by clicking on the group card views on the dashboard. This view presents a history of all payments and debts and the current debt balance for the logged-in user. Moreover, there's also an options menu in the top right corner which has three menu items, "Show members", "Add member", and "Leave group". At last, there's bottom navigation which consists of two buttons "Add payment" and "Add debt".

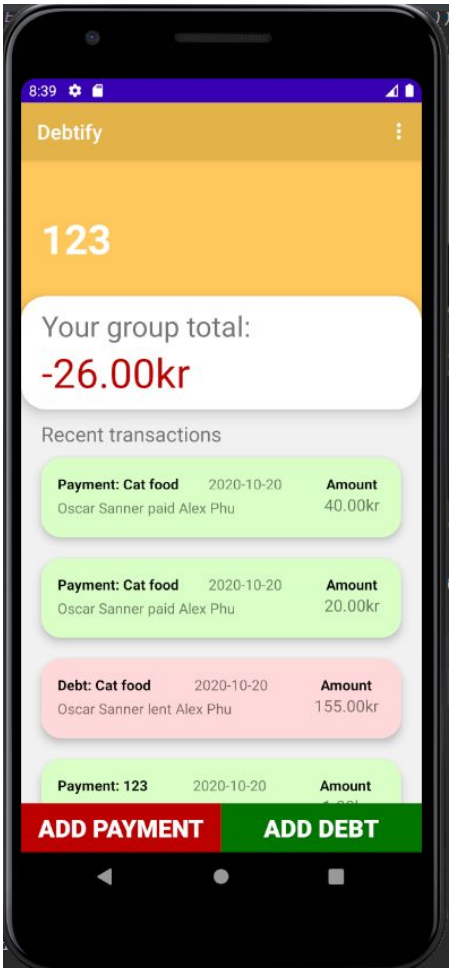


Figure 2.6: Detailed view of a group.

2.3.7 Add Debt Page

The user can navigate to the add debt page through a detailed group's bottom navigation. On this page, the user can specify who the lender and borrowers are, and whether or not the debt should be split evenly among the borrowers.

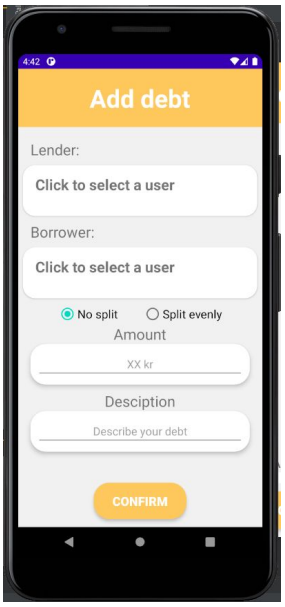


Figure 2.7: Add debt page.

2.3.8 Settle Debt Page

Similar to the “Add debt page”, the settle debt page can be accessed through a detailed group view’s bottom navigation. The user has the ability to settle any debts that are active in the group on this page.

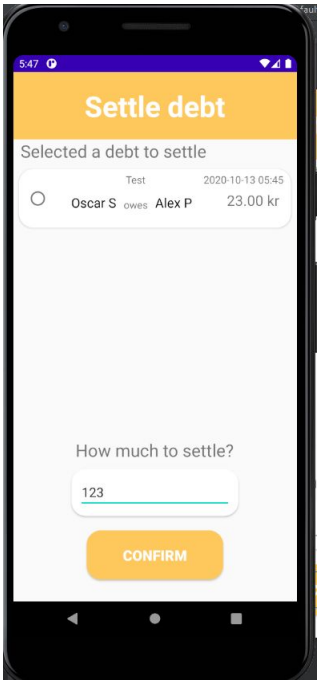
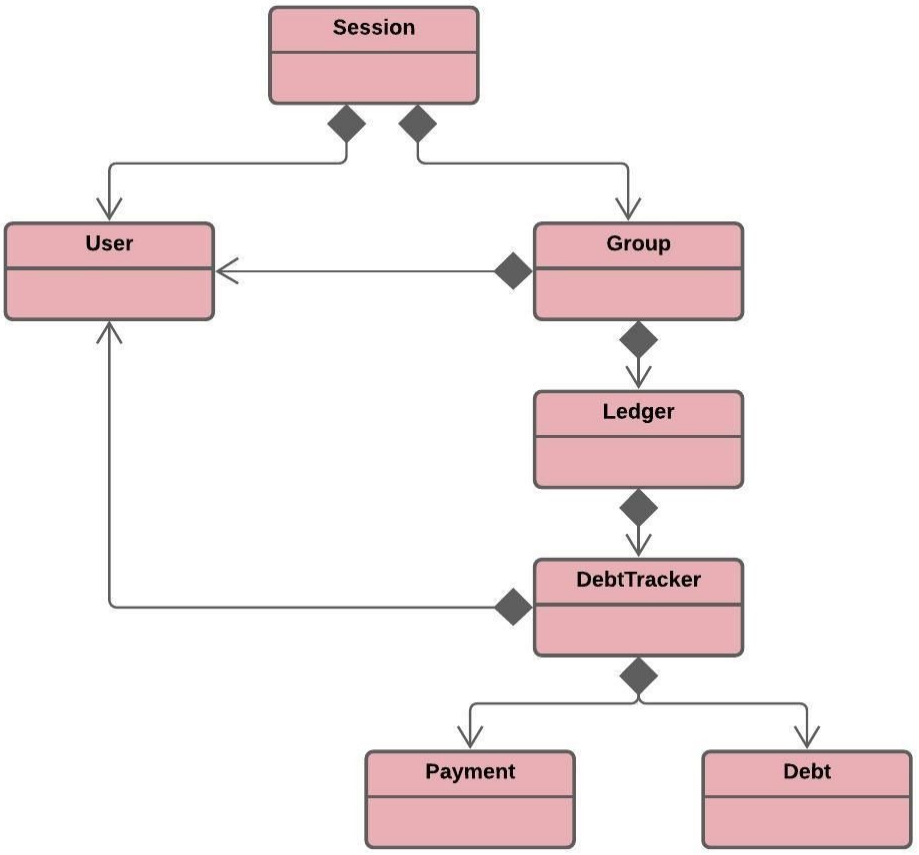


Figure 2.8: Settle debt page.

3. Domain model



3.1 Class responsibility

Session
This class manages account-level interactions. This class could be considered as the main class. It holds the logged-in user as well as acts as an aggregate for the logged-in user's details, such as groups and contacts.

User
The *User* class is, naturally, the representation for customers who are using the application. This class offers no special commands or ability to perform anything other than simply retrieving the *User* object's data.

Ledger
A ledger manages all the *DebtTrackers* in a specific group and serves as a traditional ledger. It's held by the group and organizes debts and payments.

DebtTracker

The *DebtTracker* represents the current debt balance between the lender and the borrower. Hence, this class keeps track of the details regarding the initial debt and all of the subsequent payments.

Debt
A *Debt* represents the amount of money that is owed.

Payment
A debt *Payment* represents the amount of money that is paid back.

4. References

4.1 Tools

- GitHub
  - Online version control using Git.
    - https://github.com/
- Android Studio
  - The project's IDE.
    - https://developer.android.com/studio
- Lucidchart
  - An online software for creating UML diagrams.
    - https://www.lucidchart.com/pages/
- Figma
  - A web-based platform for designing user interfaces.
    - https://www.figma.com/
- Slack
  - The project’s communication platform to discuss confidential subjects.
    - https://slack.com/intl/en-se/
- Zoom
  - For online meetings.
    - https://zoom.us/

4.2 Libraries

- JUnit
  - Java’s unit testing framework.
    - https://junit.org/junit5/
- AndroidX
  - Android’s API
    - https://developer.android.com/jetpack/androidx