Moneytor

Software Architecture Document

Version 2.0

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 26/11/2021 | 1.0 | Initial version | Team |
| 11/12/2021 | 2.0 | Add deployment view  Add implementation view | Hoàng  Thành |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[**Introduction**](#_gjdgxs) **4**

[**Architectural Goals and Constraints**](#_30j0zll) **4**

[**Use-Case Model**](#_1fob9te) **4**

[**Logical View**](#_2et92p0) **5**

[Component: Manage Spending Note](#_tyjcwt) 5

[Component: Reminder](#_dt72htqiiz1k) 7

[Component: Note on homescreen widget](#_ttf0z7a2v9fj) 7

[Component: Wallet](#_3byodnjoaodp) 9

[Component: Login using Google account](#_9bye8966pag6) 10

[Component: Sync data](#_z5uc5fe55c22) 10

[Component: Filter](#_v2bq11a8ft9o) 11

[Component: Goal reminder](#_j325vczh6fly) 12

[Component: Periodical reminder](#_g1s66miyu70b) 12

[Component: Spending visualization and analysis](#_wdbb1p330xvn) 13

[Component: Spending goal](#_ixnpbvrk33n5) 14

[Component: Manage debt](#_xvyqmn9uehzd) 15

[**Deployment**](#_3dy6vkm) **16**

[**Implementation View**](#_1t3h5sf) **16**

Software Architecture Document

# Introduction

This document provides an architectural overview of the Moneytor android application. This document presents the architecture as use case view, logical view, deployment view and implementation view. These views are presented using the Unified Modeling Language.

# Architectural Goals and Constraints

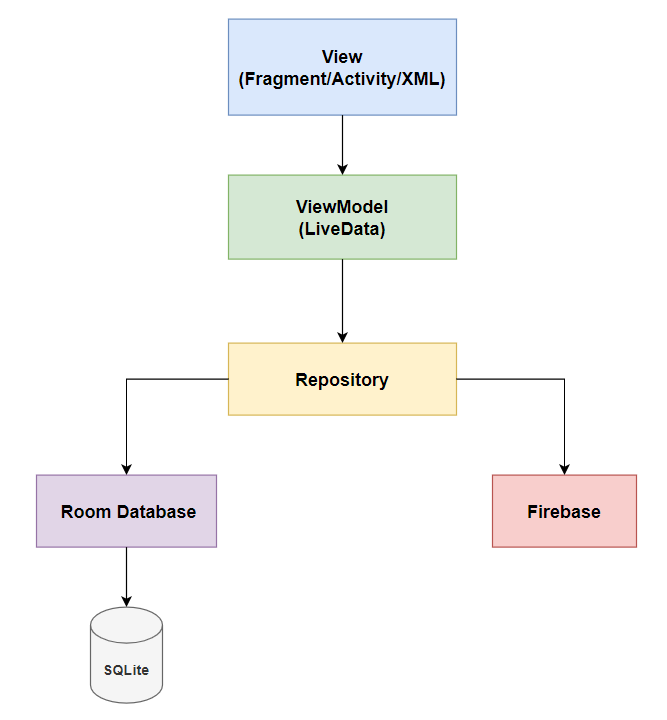
There are some key requirements and system constraint:

* Data persistence will be addressed using relational databases.
* All performance requirements, as stipulated in the Vision Document.

# Use-Case Model

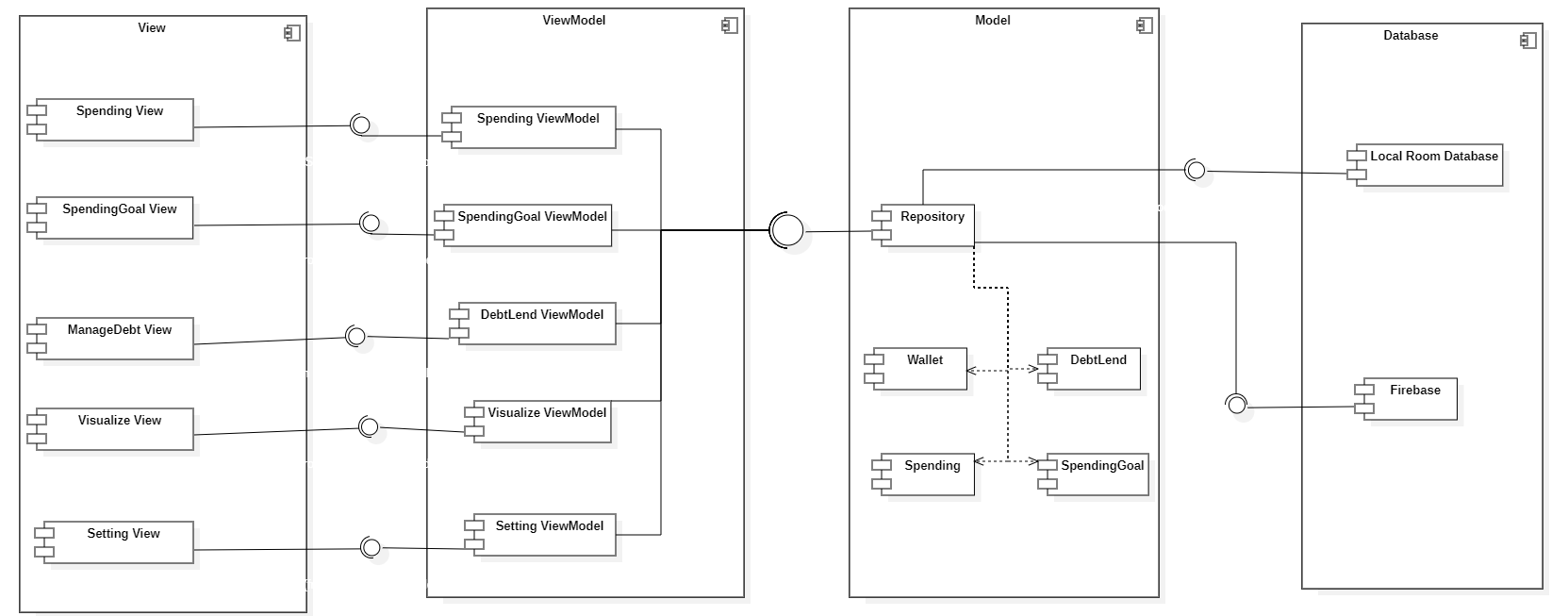


# Logical View



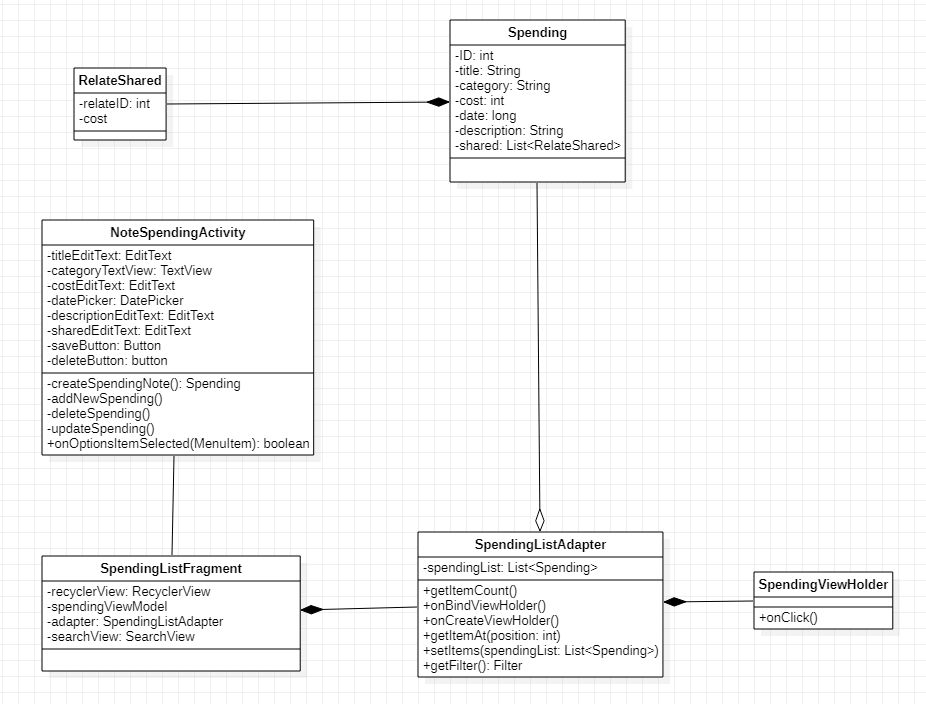
The logical view of Moneytor Android application following the Model-View-ViewModel (MVVM) architecture is that each component depends only on the component one level below it.

* **View:** contains classes for each of the UI forms that the user use to interact with the Application
* **ViewModel**: acts as a bridge between the Model and the View. It’s responsible for transforming the data from the Model and provides data streams to the View.
* **Model**: represents the data. Consists of model classes, Repository modules handle data operations, provide a clean API so that the rest of the app can retrieve this data easily, local (RoomDatabase) and remote data source (Firebase) for stored data.



## Component: Manage Spending Note

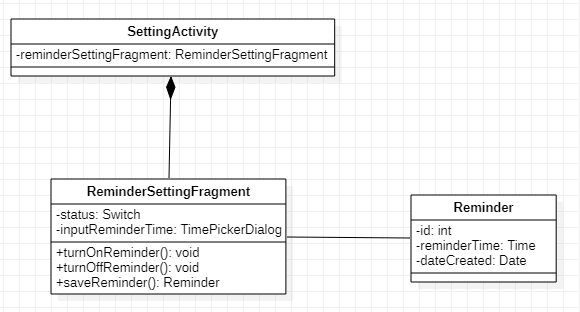
This component is responsible for all the logic related to the spending note, allowing the user to view all spending, create a new spending, delete or modify a spending.



| **Class** | **Explain** |
| --- | --- |
| Spending | Model stored data for spending note, including attributes such as ID, title of spending, category of spending, cost of spending, date created of this spending, description and a list of shared users of this spending. |
| RelateShared | Model stored data for relate shared including relateID and cost of this relate shared on a spending. |
| SpendingListAdapter | Bind data to the View holder, support recycler list view show on UI. |
| SpendingViewHolder | Display each item of the recycler list view. |
| SpendingListFragment | Display a spending note list and buttons to modify the data. It also has an option menu containing searchview to filter spending note description. Including attributes such as spendingViewModel to bind data from the database. |
| NoteSpendingActivty | Display a form support user to fill, update information in detail of a spending and write into the database or delete a spending. |

## Component: Reminder

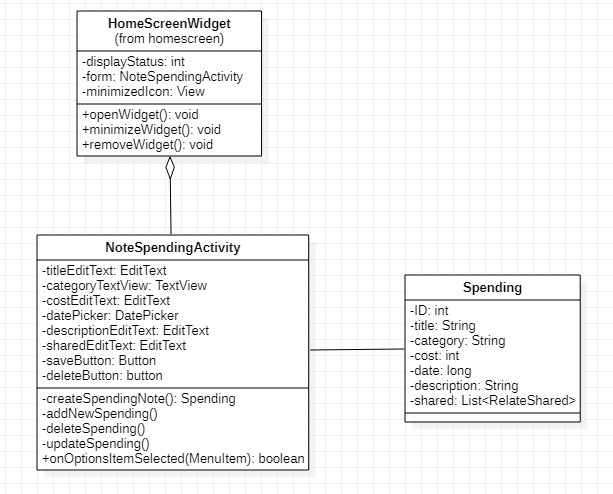
This component includes some classes related to set daily spending note reminder service such as SettingActivity, ReminderSettingFragment and Reminder class.



| **Class** | **Explain** |
| --- | --- |
| Reminder | Including attributes like ID, the time of day to reminder and the date this reminder was created, this class is used to save a set reminder. |
| ReminderSettingFragment | Including a status switch to turn on/off and a time picker dialog to pick a time of day, this class is an UI fragment that helps the user to set and save a reminder. |
| SettingActivity | This is an activity that includes Reminder Setting Fragment and some other setting fragments (which were mentioned in other components). |

## Component: Note on homescreen widget

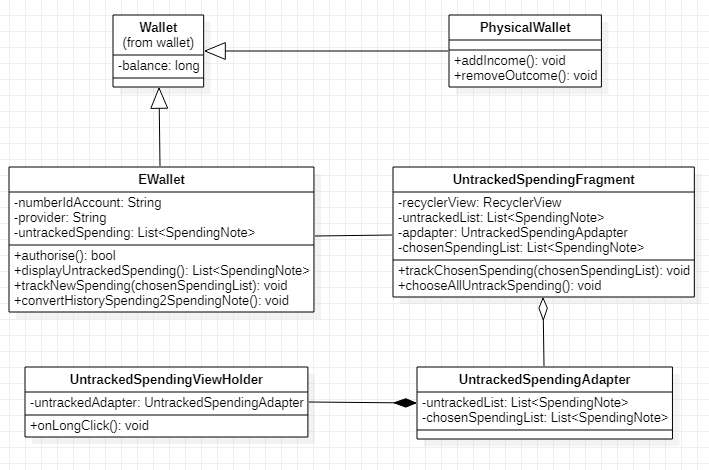
This component provides the service of note on the home screen widget.



| **Class** | **Explain** |
| --- | --- |
| HomeScreenWidget | Including the attributes like the display status of the widget (1 for open, 0 for minimize and -1 for remove), the form to note the spending, the minimized icon and the open, minimize, remove widget methods, this class describes the noting widget on the home screen and its services. |
| NoteSpendingActivity | This is the activity that displays the form for the user to note their spending in detail.  This activity class includes the input views for each information like title, category, cost, date, description, share the bill with others and implements the CRUD methods. |
| Spending | Including the attributes corresponding to each information of a spending note like ID, title, category, cost, date, description, share the bill with others, this class represents a spending note. |

## Component: Wallet

This component includes 2 types of wallet: physical wallet and e-wallet, and some related services.

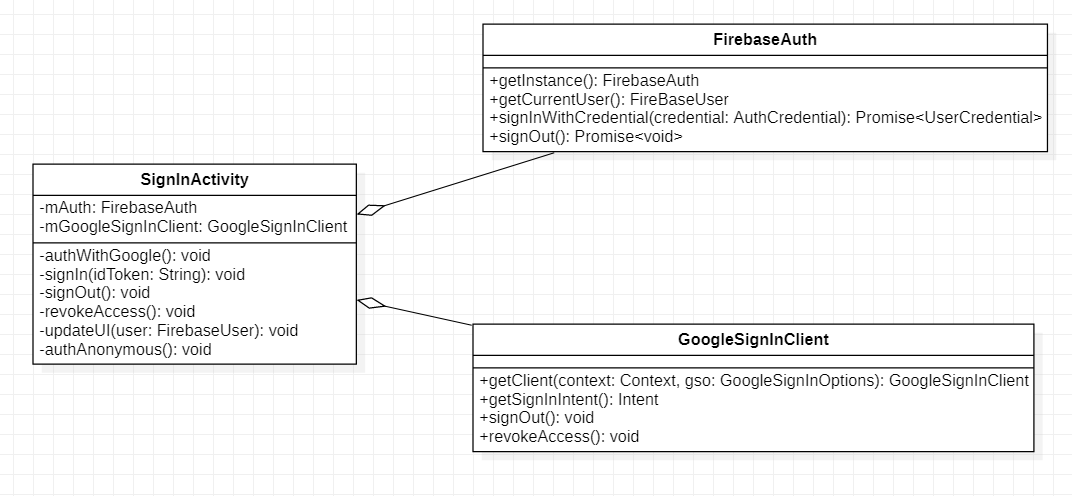


| **Class** | **Explain** |
| --- | --- |
| Wallet | Including a balance attribute, this class is extended by PhysicalWallet class and EWallet class. |
| PhysicalWallet | Inheriting the Wallet class, this class represents a cash wallet, implements 2 new methods to add the income and remove the outcome of the user. |
| EWallet | Inheriting the Wallet class, this class represents an e-wallet with some added attributes like the number ID of the e-wallet account, the e-wallet provider (Momo, ZaloPay,..) and a list of the untracked spendings that were extract from transaction history of the e-wallet. This class also supplies some services like authorising the user, displaying the untracked spending list, tracking the new spending (in the untracked spending list), and an utility method to convert the transaction history to the list of spending notes. |
| UntrackedSpendingFragment | This is the UI fragment that displays the untracked spending list extracted from e-wallet by a recycler view. This class provides some services like allowing the user to select all or some of the spending items and track them. |
| UntrackedSpendingAdapter | This is the adapter class of the recycler view in Untracked Spending Fragment. |
| UntrackedSpendingViewHolder | This is the view holder class of the Untracked Spending Adapter. The method onLongClick is overridden to select an spending note item. |

## 

## Component: Login using Google account

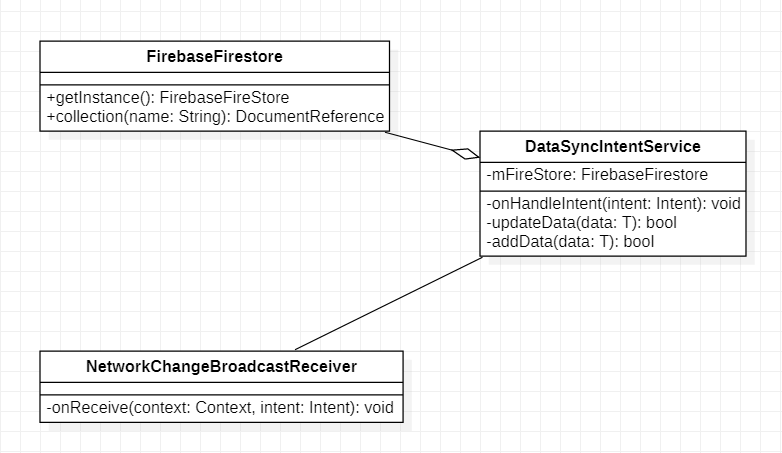
This component handles user login activities. Users can use a local account or login with a google account.



| **Class** | **Explain** |
| --- | --- |
| SignInActivity | This activity allows the user to choose their login method, either with a google account, which gives him the ability to backup his data to the cloud, or to continue with the local database. |
| FirebaseAuth | The entry point of the Firebase Authentication SDK, allows the app to use the provided authentication methods and get the current user. |
| GoogleSignInClient | A client for interacting with the Google Sign In API, where sign in and sign out methods are called. |

## Component: Sync data

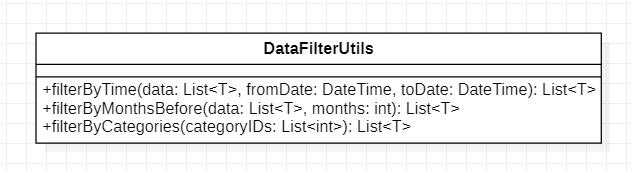
This component lets the app upload user data to the cloud using Firebase Firestore service.



| **Class** | **Explain** |
| --- | --- |
| DataSyncIntentService | A component that performs data uploading to FirebaseFirestore in background processes. |
| FirebaseFirestore | The interface for Cloud Firestore service, providing update and create data methods. |
| NetworkChangeBroadcastReceiver | Listen to events like internet connection available for automatic data backup functionality. |

## Component: Filter

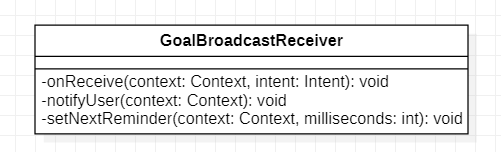
This component filters data from the database based on the criterias that the user chooses on the spending viewing activity.



| **Class** | **Explain** |
| --- | --- |
| DataFilterUtils | This class provides methods for filtering data according to criterias like time, categories. |

## Component: Goal reminder

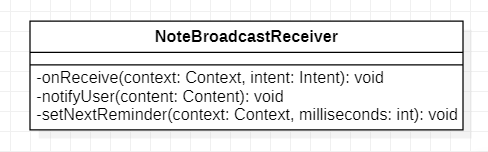
This component receives messages from the system about preset spending goals and sends user notification.



| **Class** | **Explain** |
| --- | --- |
| GoalBroadcastReceiver | Receive goal intent, and send notification to users about incoming financial goals. |

## Component: Periodical reminder

This component receives messages from the system which reminds the user to write down their spending into the app regularly.

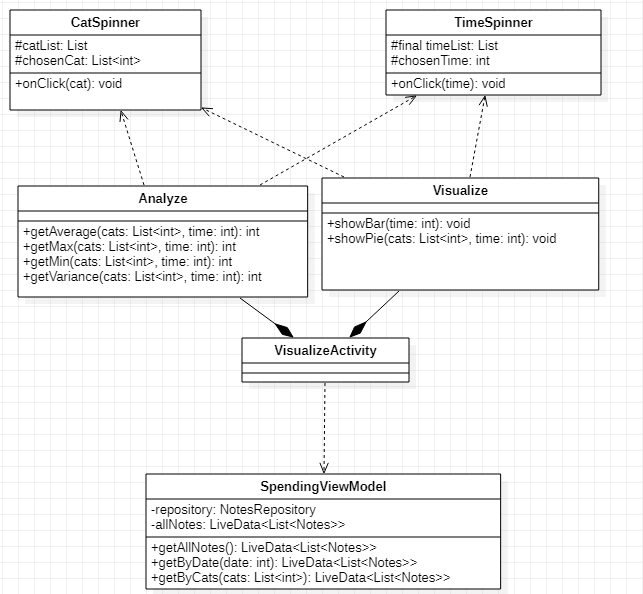


| **Class** | **Explain** |
| --- | --- |
| NoteBroadcastReceiver | Receive note taking reminders intent, then send notification reminding users to take note of their spendings. Automatically set new reminders for users. |

## 

## Component: Spending visualization and analysis

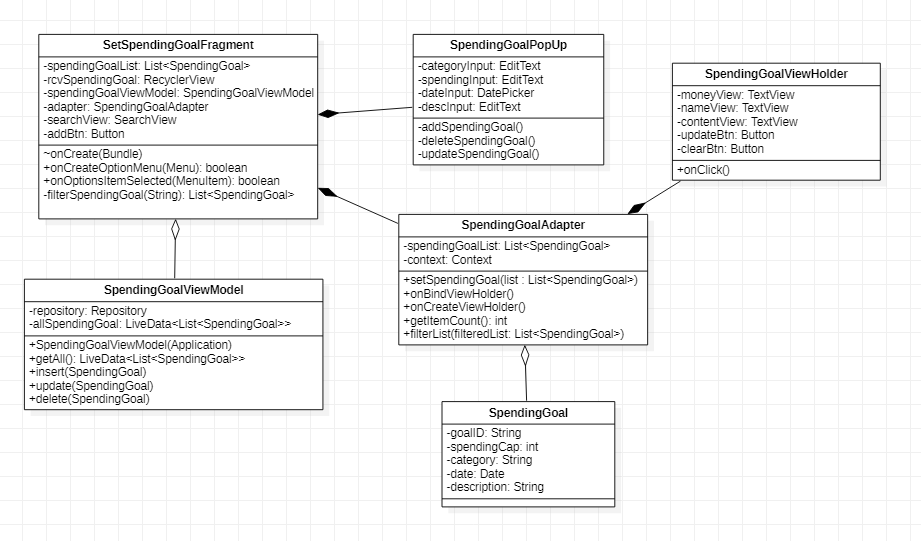
This component shows visual representations of user spendings and basic analysis.

****

| **Class** | **Explain** |
| --- | --- |
| VisualizeActivity | The activity where the visualization and analysis are set to be shown. |
| AnalyzeFragment | This class is used to calculate and display the necessary attributes of the analysis. |
| VisualizeFragment | This class is used to set the bar graph and pie graph for the detailed information of spending. |
| SpendingViewModel | This is the local data view model of the application based on the SQLite3 engine. The view model will contain a repository, of which to give the result of the queries given by the VisualizeActivity class. The view model will also save a copy of the data of the list of all spending notes in the database at creation time. |

## Component: Spending goal

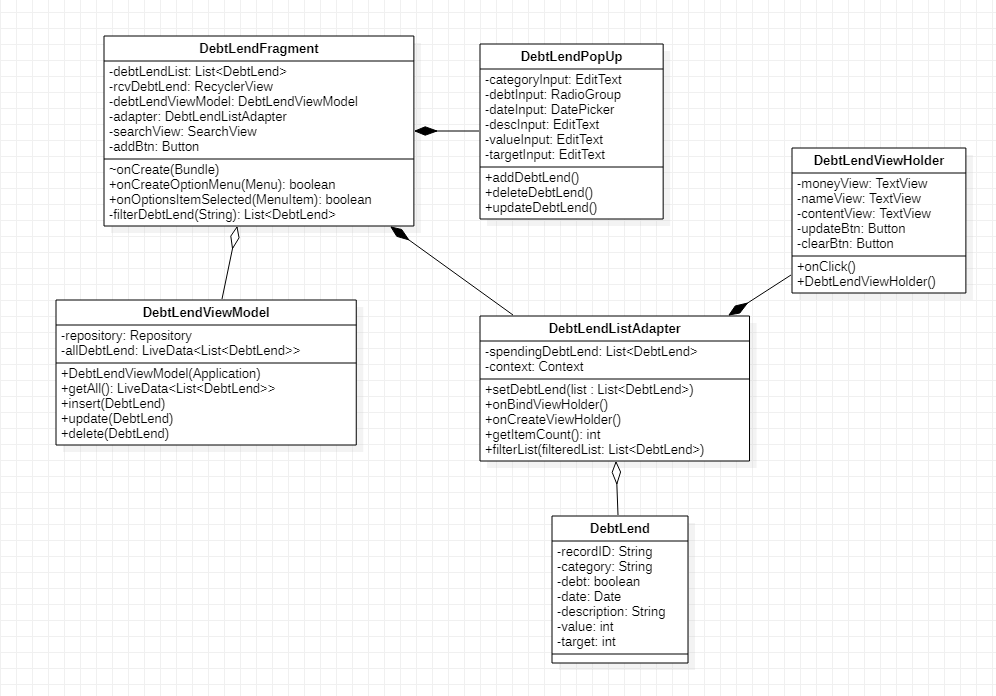
This component is used to display and add, delete, update all spending goals.



| **Class** | **Explain** |
| --- | --- |
| SpendingGoal | Contains the information of a spending goal. Each spending goal is identified by its goalID, Spending cap is the limitation of money you have to use until date. Category is null if it is just normal spending. desc is a short description about that spending goal. |
| SetSpendingGoalFragment | Display a spending goal list and buttons to modify the data. It also has an option menu containing searchview to filter spending goal description. SpendingGoalViewModel to bind data, addBtn is a button to add a new spending goal. |
| SpendingGoalPopUp | Display a popup to modify data when the user clicks on add, update button. |
| SpendingGoalViewModel | Used to access and modify databases. |
| SpendingGoalAdapter | Bind data to the viewholder. |
| SpendingGoalViewHolder | Display a recyclerview item. |

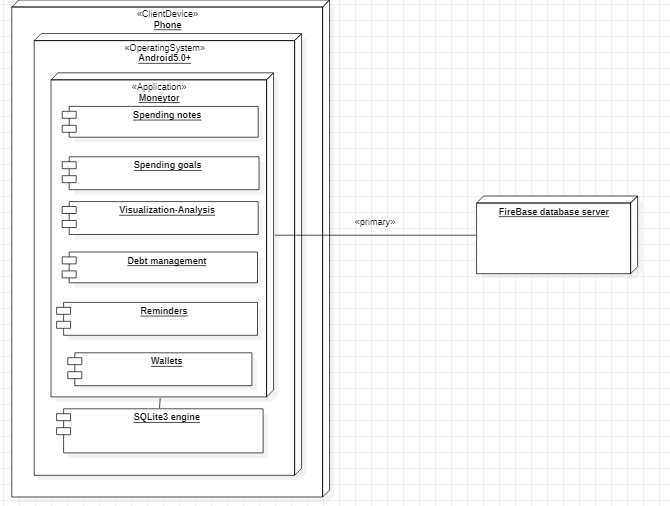
## Component: Manage debt

This component is used to display and manage debts/lends.



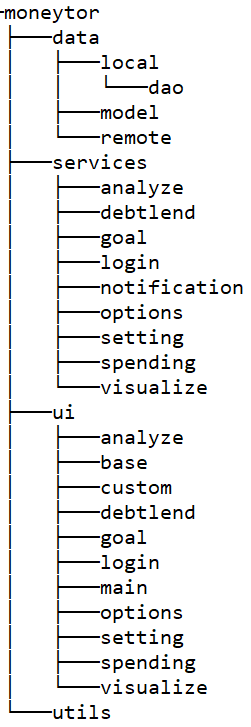
| **Class** | **Explain** |
| --- | --- |
| DebtLend | Contains the information of a debt/lend. Each debt/lend is identified by its recordID, category is category of the debt/lend, not necessary, value is the amount of money, debt has 2 values, if debt is *true* this is a lend, if debt is *false* this is a debt, target is the one you lend (debt=*true*) or lend to you (debt=*false*), date is the time of that debt/lend and desc is a short description of that debt/lend. |
| DebtLendFragment | Display a debt/lend list and buttons to modify the data. It also has an option menu containing searchview to filter debt/lend description. DebtLendViewModel to bind data, addBtn is a button to add a new debt/lend. |
| DebtLendPopUp | Display a popup to modify data when user click on add, update button. |
| DebtLendViewModel | Use to access and modify database. |
| DebtLendAdapter | Bind data to the viewholder. |
| DebtLendViewHolder | Display a recyclerview item. |

# Deployment

**

* Node 1: Phone (client device): The users’ smartphones. The client device should have Android 5.0 and up as its operating system for the Moneytor application to run properly.
* Node 2: FireBase database server: Where the external backup of users’ data, incl. spending notes, goals, reminders, etc. will be stored.

# Implementation View



**Description:**

data: include all classes relate to retrieving, storing data

* model: POJO class model
* local: Android Room Database (SQLite)
  + dao: include data access object class
* remote: Google Firebase database

services: subpackage for all background related service

* analyze: business logic for the Analyze screen
* debtlend: business logic for managing debt/lend
* goal: business logic for managing goal
* login: business logic for login
* notification: push notification
* options: business logic for select category, period, contact
* setting: business logic for setting app
* spending: business logic for managing spending note
* visualize: business logic for the Visualize screen

ui: subpackage for all UI-related packages/classes

* analyze: UI for Analyze screen
* base: base UI classes for inherited
* custom: custom view (notification, dialog, …)
* debtlend: UI for Manage Debt/Lend
* goal: UI for Manage spending goal
* login: UI for the Login screen
* main: UI for the Main screen
* options: UI for the category, period, contact list
* setting: UI for Setting screen
* spending: UI for Manage spending note
* visualize: UI for Visualize screen

utils: include all helpers supporting function class