Last edited by **Mohd Tabish Jamal** 1 year ago

ISEE 2020 | Team Technum Opus | Advanced Prototype

Bonjour! Welcome to our third blog article.

In our last blog, we discussed our first deliverable which went very well.

This time, we came up with an even better version of **MoneyLisa**. The advanced prototype is OUT..!!

In this blog, we will discuss:

- New features
- Design pattern
- Coding conventions
- Sneak peek of MoneyLisa

Let's get started.. 😃

The basic prototype was straightforward. The client asked us to include many functionalities in the advanced prototype. To give you a quick peek at it:

New Features

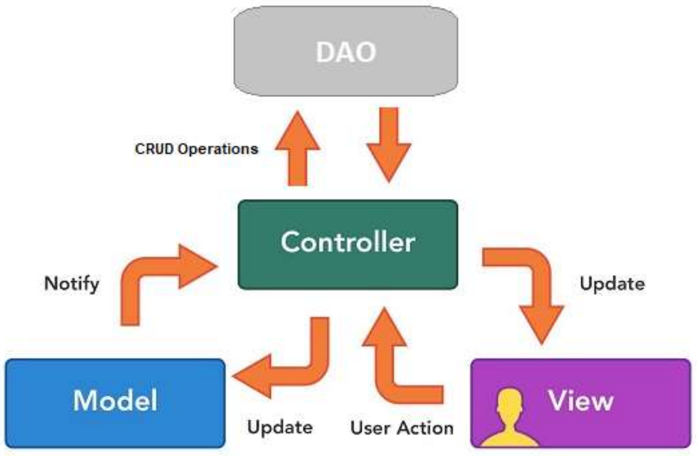
✓ ...Sign-in / Sign-up
✓ ...Multi-user Account **(*Home, Business, Travel*)**
✓ ...Borrowed Money
✓ ...Help Page
✓ ...Weekly Tips
✓ ...Budget Planning
✓ ...Currency Conversion

Design pattern

Design pattern is one of the most crucial step in app development. We will start with it's definition:

> Design patterns represent the best practices used by experienced object-oriented software developers. Design patterns are solutions to general problems that software developers faced during software development. These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

Now since we have the definition of it, you must be curious to know if we have used any design pattern for our app. The answer to this question is **YES!!** We have used a combination of two design patterns it is **Model View Controller (MVC)** along with **Data Access Object(DAO)**. So, the MVC design pattern specifies that an application consists of a data model, presentation information, and control information. The pattern requires that each of these be separated into different objects. MVC is more of an architectural pattern, but not for complete application. MVC mostly relates to the UI / interaction layer of an application. You're still going to need a business logic layer, maybe some service layer and data access layer.



- **The Model** contains only the pure application data, it contains no logic describing how to present the data to a user.
- **The View** presents the model's data to the user. The view knows how to access the model's data, but it does not know what this data means or what the user can do to manipulate it.
- **The Controller** exists between the view and the model. It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events. In most cases, the reaction is to call a method on the model. Since the view and the model are connected through a notification mechanism, the result of this action is then automatically reflected in the view.
- **The DAO** allows us to decouple the Persistent storage implementation from the rest of the application.

Why the combination of MVC and DAO..??

The reason is, MVC is easy to implement and debug. It facilitates modularity in the code, which makes code more reusable. If there is a problem in a particular module, it will not affect other modules. And in combination with DAO, it makes it very easy to separate the Persistent layer of the application. All the CRUD operations are performed in DAO layer. It efficiently separates the business logic with the database operations.

Ahh! Too many technicalities right? Well it's a technical blog 😜

Coding Conventions

Let us move forward to the second crucial part of app development that is coding conventions. We will begin with the most basic question and that is:

Why Have Code Conventions?

> - 80% of the lifetime cost of a piece of software goes to maintenance.
> - Hardly any software is maintained for its whole life by the original author.
> - Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
> - If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create -- Oracle

We have followed Java Code Conventions which was published by Sun Microsystems, Inc. in September 1997.

Because of this reason, we have applied Java Code Conventions. Some of them are listed below:

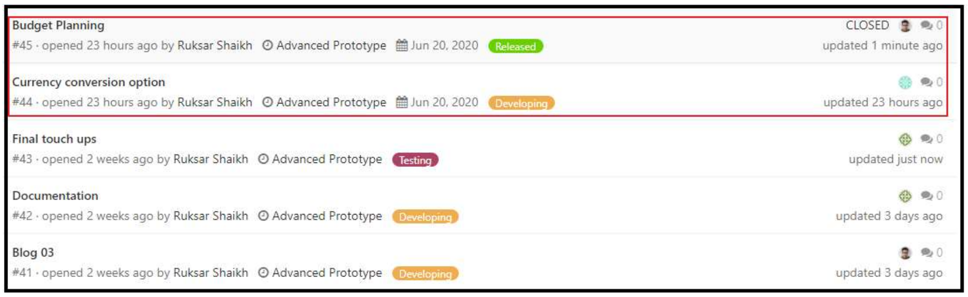
- All the **naming conventions** are standard and a developer would just read the name and understand what it is about. When defining classes, methods, or variables, *camelCase* is used. Name of class starts from Capital letter.

```
package com.technumopus.moneylisa.databaseHelper;
import android.content.BroadcastReceiver;
public class DailyRecTransBroadcastReciever extends BroadcastReceiver {
```

- **Package and Import statements** are written at the top before the class definition.
- Use of **comments** is used so that if anyone else reads this code, they should be able to understand it.
- Use of **Indentation** to make our code readable to other users, easier to edit, display how the braces match up, and show the logic of the program in an organized fashion.

Design Solution

As mentioned, we have used MVC. It's time to show you it's usage in our app and how it looks like along with our user stories. Below is the screenshot of our issues on GitLab. We will further explain the two highlighted in red.



Budget Planning
#45 · opened 23 hours ago by Ruksar Shaikh · Advanced Prototype · Jun 20, 2020 · Released — CLOSED · 0 · updated 1 minute ago

Currency conversion option
#44 · opened 23 hours ago by Ruksar Shaikh · Advanced Prototype · Jun 20, 2020 · Developing — 0 · updated 23 hours ago

Final touch ups
#43 · opened 2 weeks ago by Ruksar Shaikh · Advanced Prototype · Testing — 0 · updated just now

Documentation
#42 · opened 2 weeks ago by Ruksar Shaikh · Advanced Prototype · Developing — 0 · updated 3 days ago

Blog 03
#41 · opened 2 weeks ago by Ruksar Shaikh · Advanced Prototype · Developing — 0 · updated 3 days ago

Budget Planning

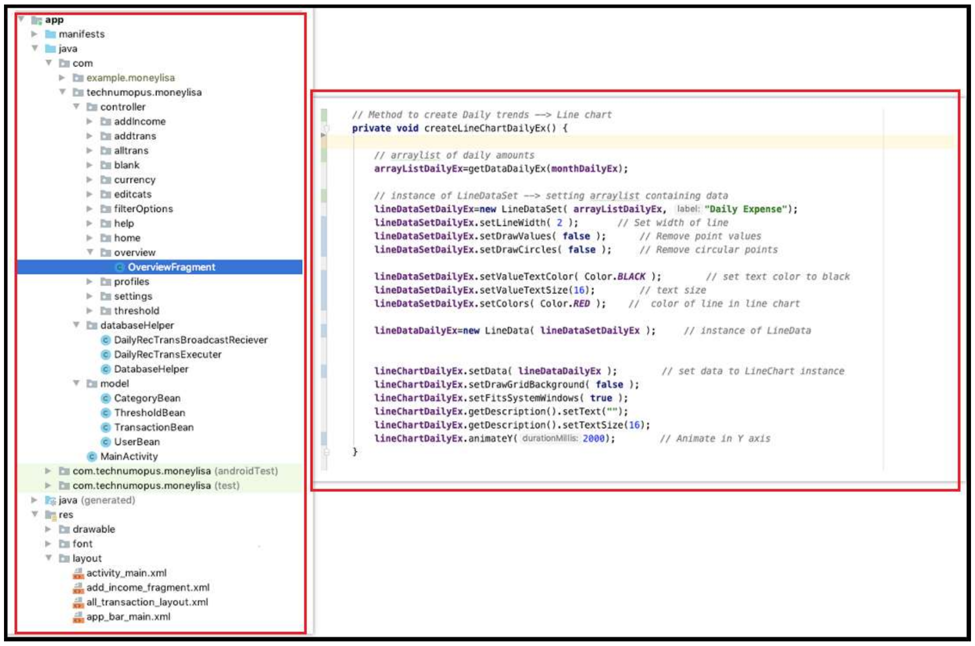
We collected requirements from the client. Based on those, we implemented the features in the App. We made changes to existing Home Fragment and created new Charts Fragment to handle this feature. On the Home Fragment, user visualizes the savings in + / - over the past 6 months. User also gets notified in case the expenses exceed the budget. In Charts Fragment, we implemented further statistical visualizations to help user take decisions regarding monthly spending.

Currency Conversion

At first, we misinterpreted this requirement. But on discussing what we implemented in first draft with the client, we came to know that we had implemented that in wrong manner. So we corrected that of course. We created a Settings Fragment where user can set default currency. Then whenever the user enters the expenses, the Spinner for currency is auto-populated to default set currency. If the user wishes to select other currency from Spinner and tries to commit the transaction, the JSON object will be fetched using Web API to convert the entered amount from selected currency to default currency. If no internet is available, user will be notified to enter the amount in default currency.

Below are the screenshots of the same from code and app:

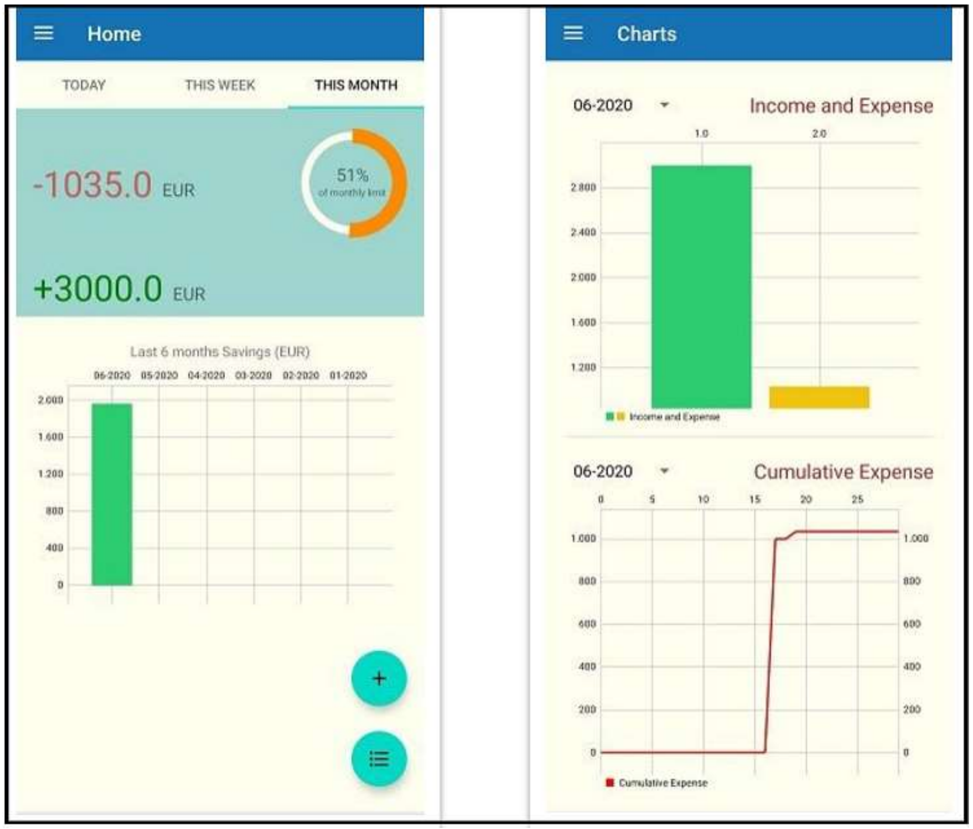
Code Snippet



```
app
  manifests
  java
    com
      example.moneylisa
      technumopus.moneylisa
        controller
          addIncome
          addtrans
          alltrans
          blank
          currency
          editcats
          filterOptions
          help
          home
          overview
            OverviewFragment
          profiles
          settings
          threshold
        databaseHelper
          DailyRecTransBroadcastReciever
          DailyRecTransExecuter
          DatabaseHelper
        model
          CategoryBean
          ThresholdBean
          TransactionBean
          UserBean
        MainActivity
    com.technumopus.moneylisa (androidTest)
    com.technumopus.moneylisa (test)
  java (generated)
  res
    drawable
    font
    layout
      activity_main.xml
      add_income_fragment.xml
      all_transaction_layout.xml
      app_bar_main.xml
```

```
// Method to create Daily trends --> Line chart
private void createLineChartDailyEx() {

    // arraylist of daily amounts
    arrayListDailyEx=getDataDailyEx(monthDailyEx);

    // instance of LineDataSet --> setting arraylist containing data
    lineDataSetDailyEx=new LineDataSet( arrayListDailyEx,  label: "Daily Expense");
    lineDataSetDailyEx.setLineWidth( 2 );       // Set width of line
    lineDataSetDailyEx.setDrawValues( false );      // Remove point values
    lineDataSetDailyEx.setDrawCircles( false );     // Remove circular points

    lineDataSetDailyEx.setValueTextColor( Color.BLACK );        // set text color to black
    lineDataSetDailyEx.setValueTextSize(16);        // text size
    lineDataSetDailyEx.setColors( Color.RED );    //  color of line in line chart

    lineDataDailyEx=new LineData( lineDataSetDailyEx );     // instance of LineData


    lineChartDailyEx.setData( lineDataDailyEx );        // set data to LineChart instance
    lineChartDailyEx.setDrawGridBackground( false );
    lineChartDailyEx.setFitsSystemWindows( true );
    lineChartDailyEx.getDescription().setText("");
    lineChartDailyEx.getDescription().setTextSize(16);
    lineChartDailyEx.animateY( durationMillis: 2000);        // Animate in Y axis
}
```

App Snippet



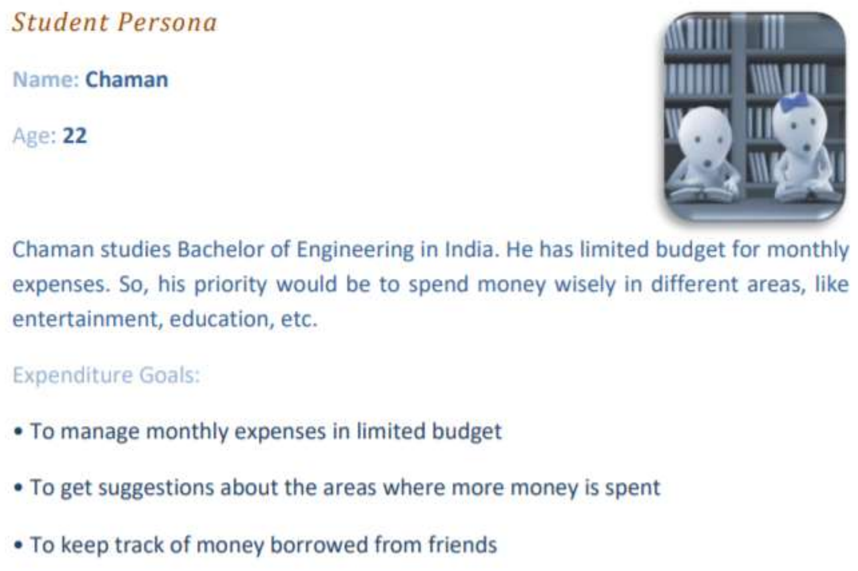
The rationale for choosing this design are its advantages and also few points as below:

- Development of the application becomes fast.
- Easy for multiple developers to collaborate and work together.
- Easier to Update the application.
- Easier to Debug as we have multiple levels properly written in the application.

Areas of usage of Money Lisa:

Now, in this part we would like to state a few of the many possible users of our application. The application will be user friendly and would find its usage with the following:

- Student (Students are always on a budget be it any age group. So they will find this application really handy and full of use to track their spending and with budget analysis, save their money too!
- Working class people (The application would also find its way to the working class people to again track their expenses along with their saved income and analyse.



Student Persona

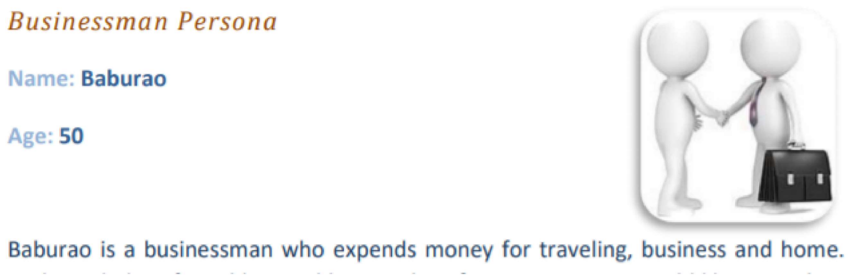
Name: **Chaman**

Age: **22**

Chaman studies Bachelor of Engineering in India. He has limited budget for monthly expenses. So, his priority would be to spend money wisely in different areas, like entertainment, education, etc.

Expenditure Goals:

- To manage monthly expenses in limited budget
- To get suggestions about the areas where more money is spent
- To keep track of money borrowed from friends



Businessman Persona

Name: Baburao

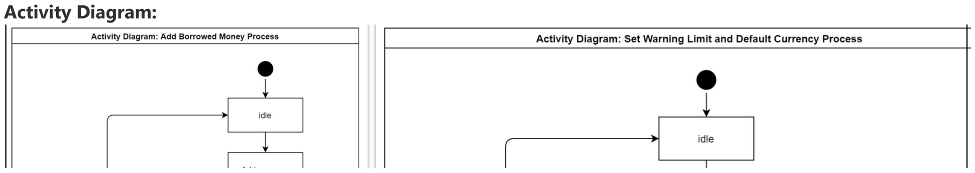
Age: 50

Baburao is a businessman who expends money for traveling, business and home.

Diagrams

Accommodating the new features/changes in our ongoing project was a bit challenging and it changed our previously made structural and behavioral diagrams. The changed diagrams are as below:

Activity Diagram:


Activity Diagram: Add Borrowed Money Process
idle
Activity Diagram: Set Warning Limit and Default Currency Process
idle

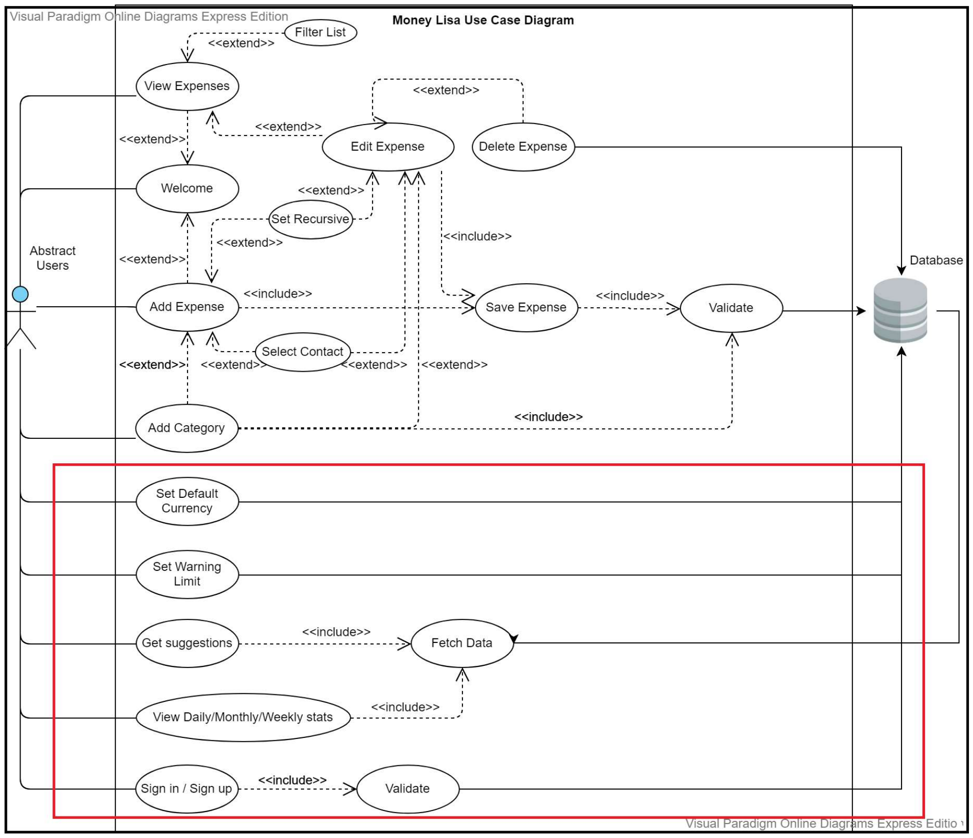
Add Borrowed Money Process:

User can add expense if the money is borrowed from some friend or relative. User fills up all the expense entries. And, when user clicks Borrowed From text box, the phone contact list will open. There the user can select the contact from whom the money is borrowed. User can set the category to be Loan. Also, optionally, there is a standard feature of making it a repetitive expense. Once all the options are set, the validation pipeline will be executed on all the values in UI. Successful validation results in expense committed in database.

Set Warning Limit and Default Currency Process:

User can navigate to Settings. There, the user can fill entries like Warning Limit or Default Currency. Once set, as the user clicks Save, the values are committed in Database. The validation of values is taken care of in UI itself, so no validation would be needed here.

Use Case Diagram:



Upper half part is our Basic Prototype. Lower half shows the use cases we implemented in Advanced Prototype.

• User can set **default currency** and **monthly warning limits** in Settings. Based on this, user gets notifications if any time in a month, the total expense exceeds 50%, 75% or 100%. User also **gets suggestions in notification** if more money is spent in certain categories like “Entertainment” or “Dine Out”.

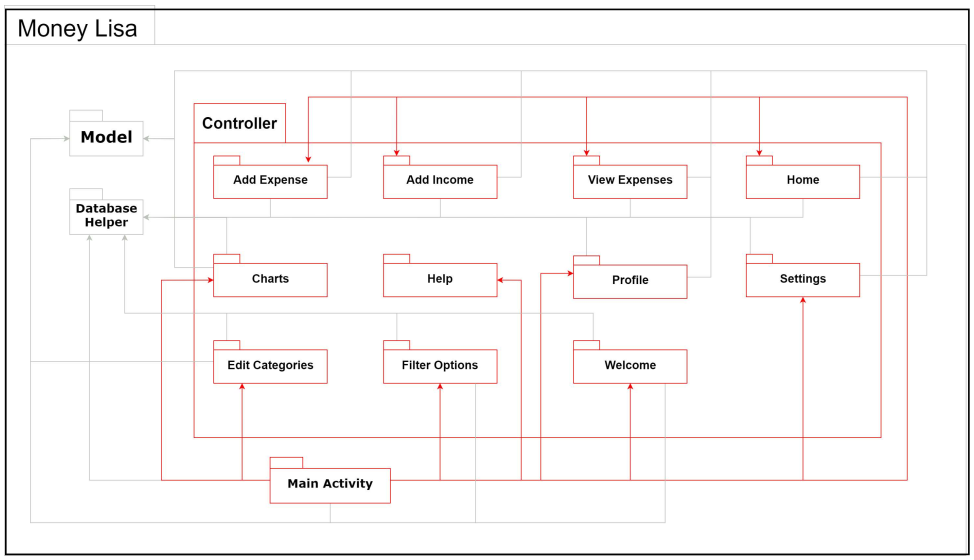
• Setting default currency allows user to enter the expense in any predefined currency, which gets converted to default currency set by user. Conversion takes place by getting JSON object from Web API service for **currency conversion**. In case of no internet, user will be notified to enter the amount in base currency only.

• **Getting suggestions** is the feature the user gets based on monthly limits and target savings set by user.

• User also gets to see daily, weekly or **monthly overview** of expenses and incomes on the Home Page itself. User can also check the **last 6 months savings** in a chart on Home Page.

• **Sign-in Sign-up** feature ensures authentication of user, which is strictly validated in Database. Password standards are also strict.

Package Diagram:

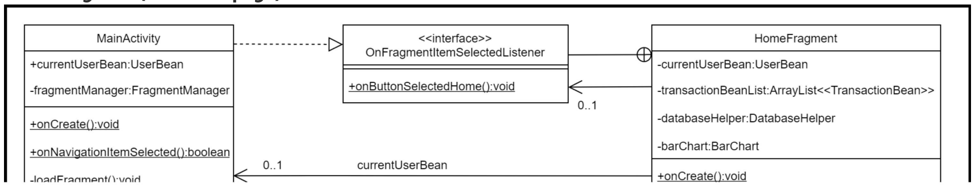


• Here a particular state of the Package Diagram is highlighted in red. Main Activity is associated with all the controller packages. Each of the controller package corresponds to the Fragment (Page/View) of an App. Main Activity orchestrates the flow of information from one Fragment to another. Main Activity also listens to the events in all the fragments.

• Also, all the packages (Except Help Page) are associated with Database as they need to fetch or commit data.

• All the packages (except Help Page) are associated with Model package. Model package basically defines the data structure of Entries, Users and Categories (corresponding to which there are tables in Database).

Class Diagram: (For Homepage)


MainActivity
+currentUserBean:UserBean
-fragmentManager:FragmentManager
+onCreate():void
+onNavigationItemSelected():boolean
-loadFragment():void
<<interface>>
OnFragmentItemSelectedListener
+onButtonSelectedHome():void
0..1
HomeFragment
-currentUserBean:UserBean
-transactionBeanList:ArrayList<<TransactionBean>>
-databaseHelper:DatabaseHelper
-barChart:BarChart
+onCreate():void
0..1
currentUserBean

• Here only the part of class diagram is shown (all the attributes and methods are not mentioned for simplicity of explanation).

• This portion of class diagram explains how the Chart and values show up on Home Page. Main Activity implements Listener Interface (which is in composition with Home Fragment). The reason is, Main Activity must listen to some events in Home Page which might lead to go to other Fragment (like say Settings Fragment).

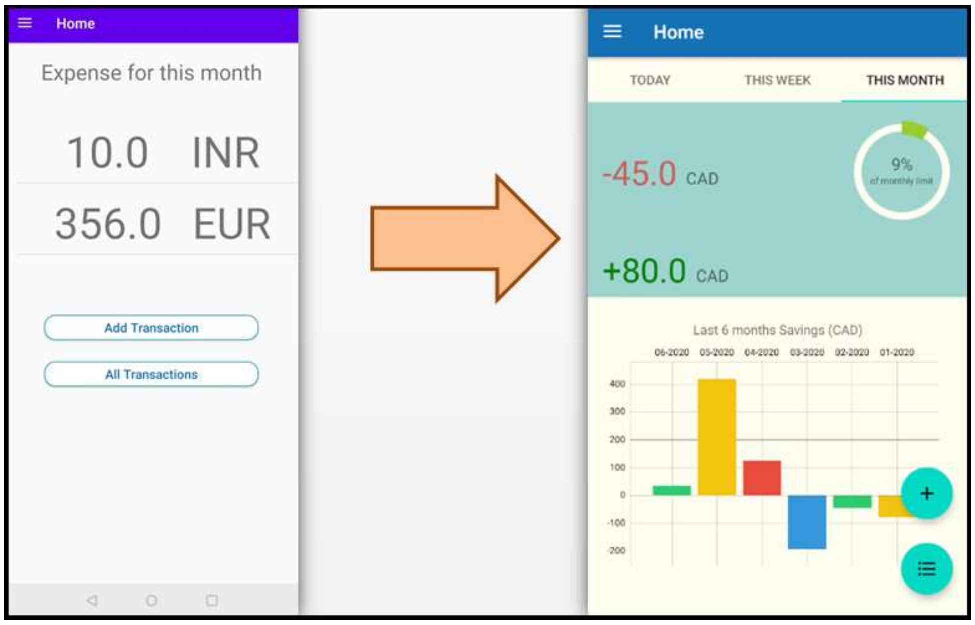
• Main Activity holds the object of current user (when user signs up or signs in). Home Fragment has association with Main Activity for calling that current user object. Using this, Home Fragment communicates with Database Helper with 0..1 multiplicity (because Database Helper single object has enough functionalities). Home Fragment is also associated with Transaction Bean Model because all we need to display certain data on Home Page is a "List of Transactions" (intuitively, a single user might have many transactions in a month or week). This is the reason, the multiplicity is 0..*.

Please find the below link for our **complete class diagram:**

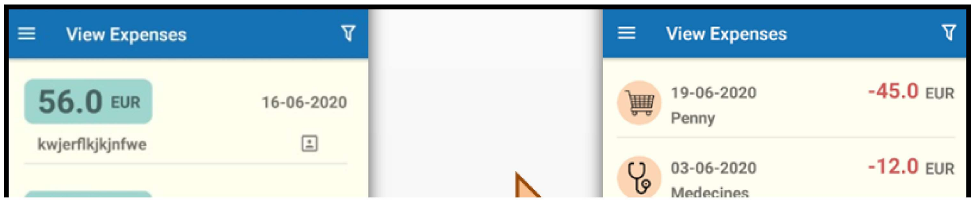
https://code.ovgu.de/steup/technum-opus/-/tree/patch-1/Diagrams/AdvancedPrototype/Class_Diagram

We also made certain **UI changes** as below:

UI Design Change for Homepage



UI Design Change for View Expense Page



Now let's move to the most exciting part of advanced prototype. That is the snippets of the app and the apk 😃

Screenshots and APK

Sign-in, default currency and profiles

Charts

Homepage and notification

Modified view expenses and help page

APK

moneylisa_v02.apk

This Brings us to the end of our blog. We will see you in our next blog article with beta prototype. Adios!

References:

https://techterms.com/definition/mvc

https://www.photocollage.com/

https://app.diagrams.net/

https://www.tutorialspoint.com/design_pattern/index.htm

https://www.oracle.com/technetwork/java/codeconventions-150003.pdf