Last edited by **Ruksar Shaikh** 1 year ago

ISEE 2020 | Team Technum Opus | Beta Prototype

Hallöchen! Welcome to our fourth blog article.

In our last blog, we discussed our advanced prototype which was successful. In today's blog, we are going to discuss the testing methodology which we used to test Money Lisa and few add on functionalities.

In this blog, we will discuss:

- Testing Methodology
- Implementation
- Summary of changes
- Sneak peek of MoneyLisa

Let's get started.. 😃

When we made the app, we looked into the design pattern and the coding conventions to be used. Today we are going to test them to see if they work as expected or not.

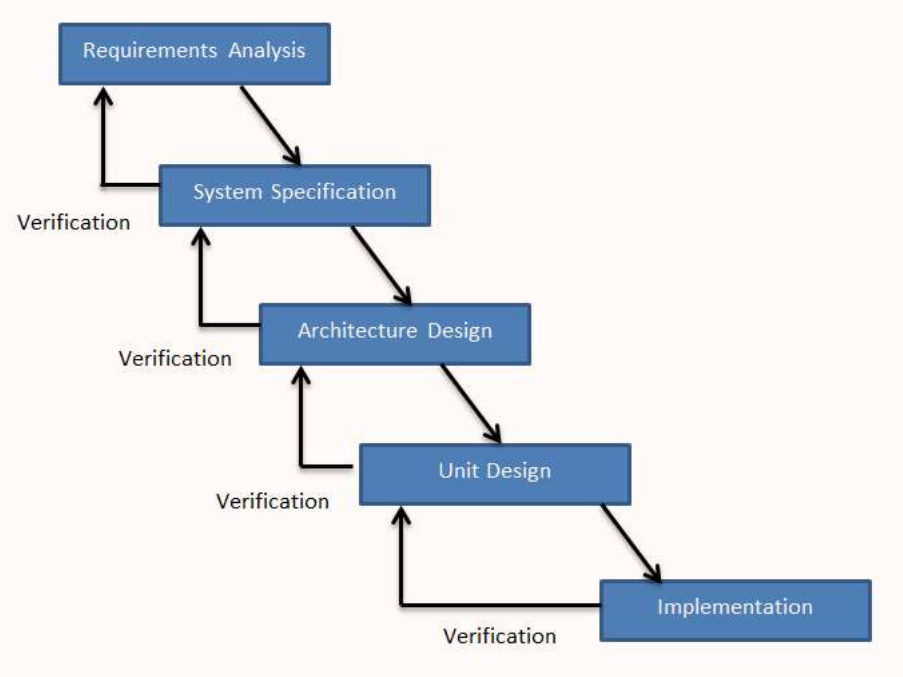
Testing Methodology

We maintained an excel sheet with all the possible testing scenarios and worked through with it.

S.No.	Test Description	Verification/Validation	Testing Phase	Blackbox/Whitebox	Failure (Yes / No)	Failure Description	Fault	Error
21	Testing editing and deleting the transactions reflects in database and transactions list	Validation (Dynamic)	Integration	Blackbox	No			
22	Testing the correctness of data displayed on charts	Validation (Dynamic)	Integration	Whitebox	Yes	Categorywise chart is congested	Chart is not horizontally scrollable	Difficult to use the MP Charts API
23	Testing that on raising the issue, the mail gets sent with appropriate data	Validation (Dynamic)	Unit	Blackbox	No			
24	Testing logout	Validation (Dynamic)	Unit	Blackbox	No			
25	Testing the correct category icons show in transactions list	Validation (Dynamic)	Unit	Blackbox	No			
26	Testing that newly added categories stay specific to the User	Validation (Dynamic)	Integration	Blackbox	No			
27	Testing that Permissions Denied scenario is handled	Validation (Dynamic)	Integration	Whitebox	Yes	If user denies the contacts permission, user will not be informed while trying to select the contact	In add expense fragment, there is no condition that if permission denied, then the behaviour of some UI needs to be different.	Such a circumstance did not strike to the mind
28	Testing the App in Landscape mode	Validation (Dynamic)	Integration	Blackbox	No			
29	Testing the Model methods	Validation (Dynamic)	Unit	Whitebox	No			
30	Basic Prototype testing	Validation (Dynamic)	System	Blackbox	No			
31	Advanced Prototype Testing	Validation (Dynamic)	System	Blackbox	No			
32	Beta Prototype Testing	Validation (Dynamic)	System	Blackbox	No			
33	Testing name conventions	Verification (Static)	NA	NA	Yes	Name of model classes may not be as per convention	Suffix "Bean" does not seem to be the right word choice	Misunderstanding of Naming of Model Classes
34	Testing access modifiers	Verification (Static)	NA	NA	No			
35	Testing appropriate time space efficiency	Verification (Static)	NA	NA	No			
36	Testing reusability	Verification (Static)	NA	NA	No			

As seen from the image, we did **Static and Dynamic Testing**.

In static testing, also called as **verification**, we tested the naming conventions, access modifiers, time space efficiency and re usability. This was done without executing the code. We did Static Testing at the end of every design phase. This helps us ensure that we are building the product right. Below is the workflow of how the static testing was performed after every milestone.

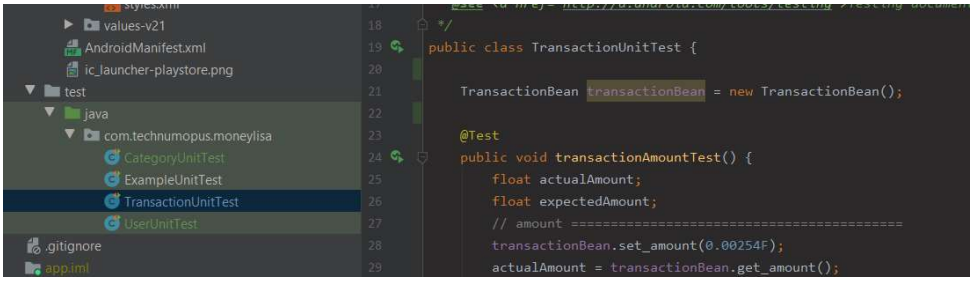


Then comes dynamic testing, in this type of testing, we compared the actual behavior of the application to the expected behavior. In other words, working with the system with the intent of finding errors.

Hence dynamic testing is a process of **validation** of software applications as an end user under different environments to build the right product. In Dynamic Testing, we did Unit Tests, Integration Tests and System Tests. Integration Tests were Blackbox / Whitebox or both, depending on the requirements. The MVC design Architecture proved to be very helpful in doing Unit and Integration Tests. Most of the Dynamic Tests were Topdown, except for some of the testings like of Recursive Transactions which was Bottomup.

Overall, while testing, we also designated Failure, Fault and Error.

We did **semi automatic testing** as shown below, for the Unit Tests of Model classes like Transaction, User and Category. We implemented this using Java Unit Test classes and Mockito. We mocked the Model Objects, set various values in them and validated those values as per requirements analysis.



☞ **Which types of failures and consequently errors did we find?** Well the answer to this is shown in the below figure:

Test	Failure	Fault	Error
Testing the correctness of data displayed on charts	Categorywise chart is congested	Chart is not horizontally scrollable	Difficult to use the MP Charts API
Testing that Permissions Denied scenario is handled	If user denies the contacts permission, user will not be informed while trying to select the contact	In add expense fragment, there is no condition that if permission denied, then the behaviour of some UI needs to be different.	Such a circumstance did not strike to the mind
Testing name conventions	Name of model classes may not be as per convention	Suffix "Bean" does not seem to be the right word choice	Misunderstanding of Naming of Model Classes
Testing the list of expenses shows correct data	There was a wrong icon for income related entries in the list fragment	"IF" condition for the income related recycler view item was not set to show correct icon	Missed by developer
Testing the list of expenses shows correct data	The icon for repititive expenses was not showing in the list fragment	"IF" condition for the recursive entries related recycler view item was not set to show correct icon	Missed by developer
Testing that correct data shows on Home Page	Entried entered on Sunday were not reflecting in "Week" tab	The function which finds all the dates of current week was having a problem, due to which, the current week dates loop was not functioning if current day is Sunday	Logical error by developer
Testing Flow of App from one fragment to another	Back button was not working for Charts and Settings Fragment	"IF" condition for the same in Main Activity was absent	Missed by developer
Testing the App in Landscape Mode	App was failing in landscape mode	Android Manifest option was not added	Such a circumstance did not strike to the mind
Testing the App in Landscape Mode	All the fragments were getting cut and not scrolalble	ScrollView was not at the Higher Level to Constraint Layout in the XML file of parent fragment	Such a circumstance did not strike to the mind

Implementation

Influence of Testing on Implementation

We maintained the backlog of open Tests in Excel. The SCRUM Master distributes the responsibilities for all tests among all the team members. It was done considering the concurrency in mind, because 2 members cannot work on the same package at the same time to avoid merge conflicts in GitLab.

We then individually looked after solving the failure in all open tests. Many a times, implementation work flow changed as in adding extra condition or removing some conditions, changing looping algorithm, changing some line of codes, etc.

Below is one of our example when we faced problem in Currency Conversion:

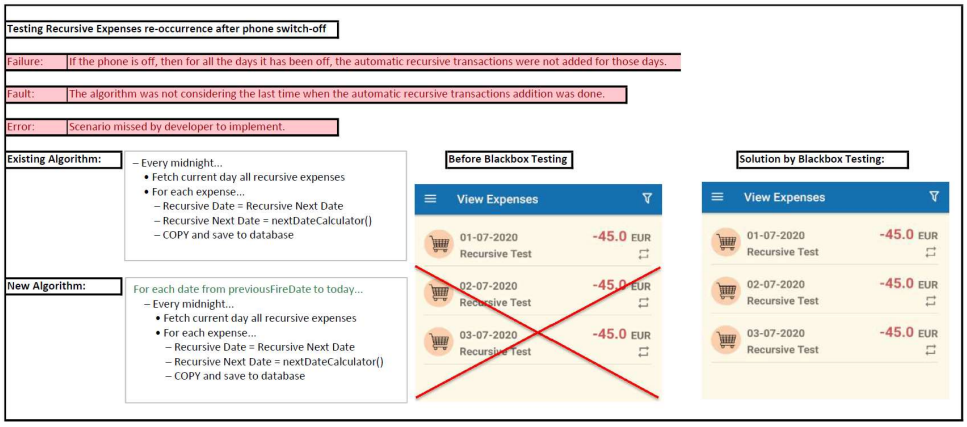
```
private float getCurrencyFactor(String currencyFrom, String currencyTo){

    String str = "abc";
    try {
        URL url = new URL( spec: "https://api.exchangeratesapi.io/latest?base=" + currencyFrom + "&symbols=" + currencyTo);
        URL url = new URL("https://free.currconv.com/api/v7/convert?q=" + currencyFrom + "_" + currencyTo + "&compact=ult
        BufferedReader br; //jk19061350
          https://api.exchangeratesapi.io/latest?base=USD&symbols=EUR
        if (isOnline() == true) { //jk19061350
            br = new BufferedReader(new InputStreamReader(url.openStream())); //jk19061350
            str = br.readLine(); //jk19061350
        } //jk19061350
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    str = str.replaceAll( regex: "[^\\d.]", replacement: "");
    float finalValue = -99;
    try {
        finalValue = Float.parseFloat(str);
        System.out.println(finalValue);
    } catch (Exception ex) {

    }
    return finalValue;
}
```

Initially, we used ***free.currconv.com***, but it had some problems, due to which we shifted to using ***api.exchangeratesapi.io***. So, in implementation, we removed the line pointing to the lower arrow and added the line pointing to the upper arrow. But, of course, there was some time spent in brainstorming with the client and team members, and also statically checking that just by replacing this line, the function will work.

👉 Example of test usage



Test case scenario:

- Transaction was added on 1st July in the Emulator.
- On 2nd July, the Emulator was not started.
- On 3rd July, the Emulator was started.
- This way, the Phone Switch Off scenario was simulated.
- After changing the Algorithm, 1st July Transaction was copied to 2nd and 3rd July.
- All of them were visible to User in the list, even though User switched off the phone on 2nd July.

UI changes made due to testing

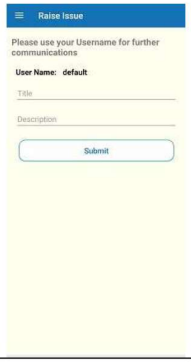
App crashes in landscape mode

When the app was being tested, we found out that the app keeps on crashing in the landscape mode as shown below:

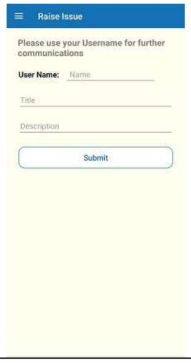
-> The option was added in Android Manifest to handle landscape mode
-> All the Fragments were added Scroll View on top of Constraint Layout.
-> UI elements widths/heights were set to "match_parent" and "wrap"content" wherever suitable

Raise Issue; guest user name was not sent in Email

In raise issue, when the user is signed in as guest, the username assigned to him/her is default.



When we contact the users, they should have a unique username. So we corrected this and added a textview for them to include their username as below:



Coding changes made due to testing

Testing the Currency Conversion

Failure: User was not able to enter the amount in different Currency

Fault: Currency Conversion API free limit was expired

Error: API Documentation was not referred correctly, which lead to this fault

Existing API: free.currconv.com

New API: api.exchangeratesapi.io

NewCode

```
try {
    URL url = new URL( spec: "https://api.exchangeratesapi.io/latest?base=" + cur
    URL url = new URL("https://free.currconv.com/api/v7/convert?q=" + currency
    BufferedReader br; //jk1986135
```

Approach:

Criteria for Equivalence Class		
1A	source currency	is default
2A	source currency	is not default
1B	source amount	is > 0
2B	source amount	is <= 0
1C	source amount le	is > 10
2C	source amount le	is <=10

Test Cases	Input - Amount	Input - Currency	Result
1A,1B,1C	45678451374	EUR	1 Error: "amount cannot exceed 10 digits"
1A,1B,2C	178	EUR	in Database: source_currency=default, source_amount=x, currency=default, amount=x
1A,2B,1C	-45784651436	EUR	not possible: UI does not accept negative amount
1A,2B,2C	-984	EUR	not possible: UI does not accept negative amount
2A,1B,1C	45678451374	INR	1 Error: "amount cannot exceed 10 digits"
2A,1B,2C	178	USD	in Database: source_currency=not_default, source_amount=x, currency=default, amount=y
2A,2B,1C	-45784651436	AUD	not possible: UI does not accept negative amount
2A,2B,2C	-984	CAD	not possible: UI does not accept negative amount

Testing the Add Expense process

Failure: Once the User adds the contact, User was not able to remove the contact.

Fault: In the implementation, there was no code to make the borrowed contact field blank.

Error: Scenario missed by developer to implement.

Existing Code:

```
480
481
482
483
        case R.id.addtrans_et_borrowed:
            EditText et =v.findViewById(R.id.addtrans_et_borrowed);
            doLaunchContactPicker(v);
            break;
```

NewCode:

```
481
482
483
484
485
        case R.id.addtrans_et_borrowed:
            EditText et =v.findViewById(R.id.addtrans_et_borrowed);
            et.setText("");
            doLaunchContactPicker(v);
            break;
```

Summary of changes

We added two new requirements given by the user and they are:

- Export option
- Raise an issue/Contact support form

Let's have a sneak peek at our user stories. Below is the screenshot of our issues on GitLab. We will further explain the two highlighted in red.

We collected requirements from the client. Based on those, we implemented the features in the App.

Export option

The user should be able to export the data in the form of excel sheet or csv file from the application. So we enabled this requirement in our app MoneyLisa.

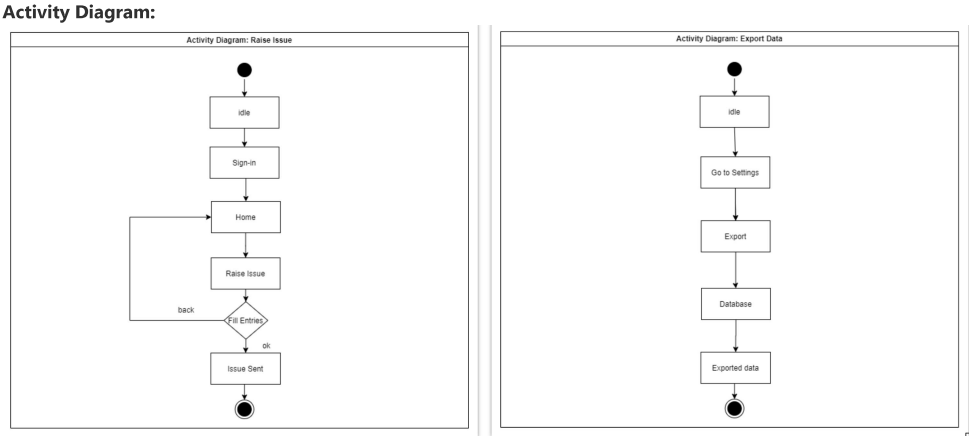
Raise an issue/Contact support form

The user should be able to contact the developers in case of an issue, so for this, we created a separate section to raise an issue faced by the user which cannot be solved through the help menu.

Diagrams

Accommodating the new features/changes in our ongoing project lead to changes in the previously made structural and behavioral diagrams. The changed diagrams are as below:

Activity Diagram:



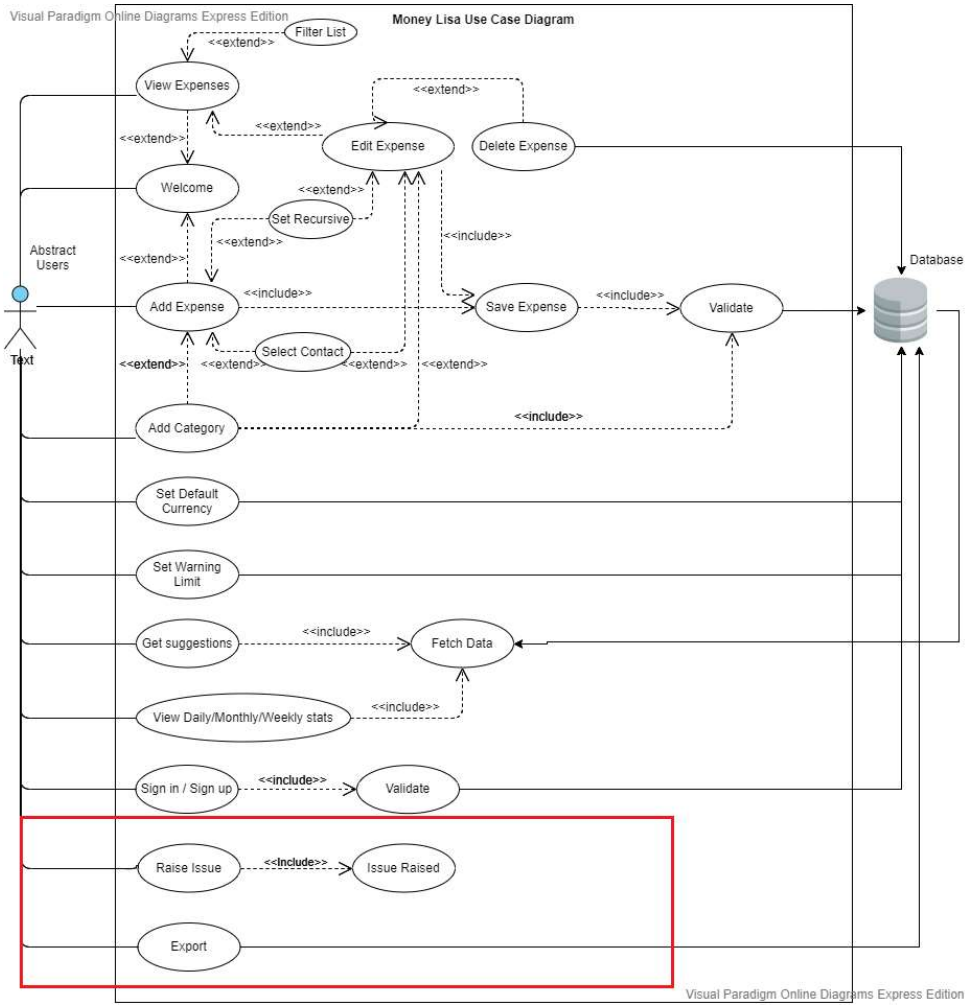
Export option

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.

Raise an issue/Contact support form

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:

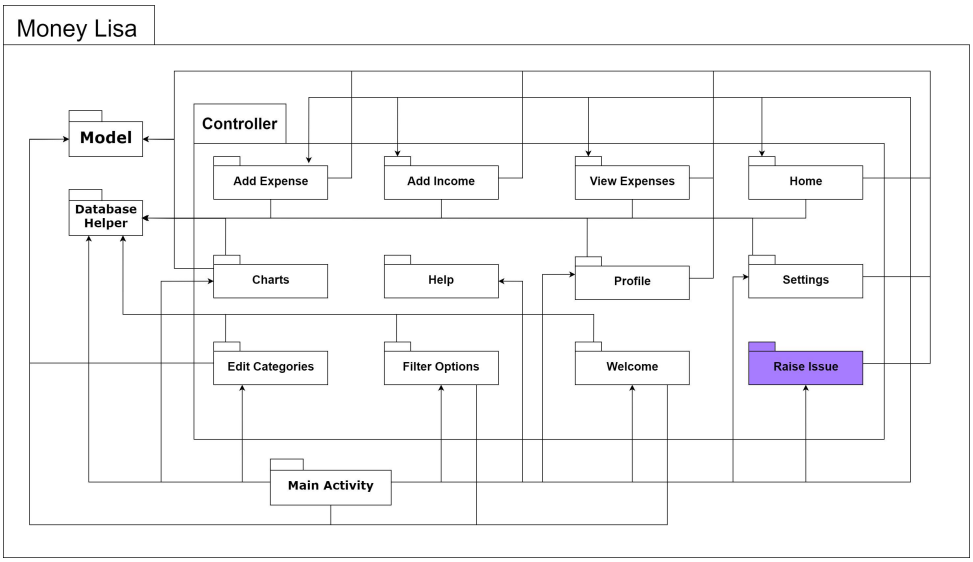


Upper half part is our Advanced Prototype. Lower half highlighted in red rectangle shows the use cases we implemented in beta Prototype.

• User can export his/her data in the form of an excel sheet or csv (comma separated value) file. This gives user to save his/her records permanently

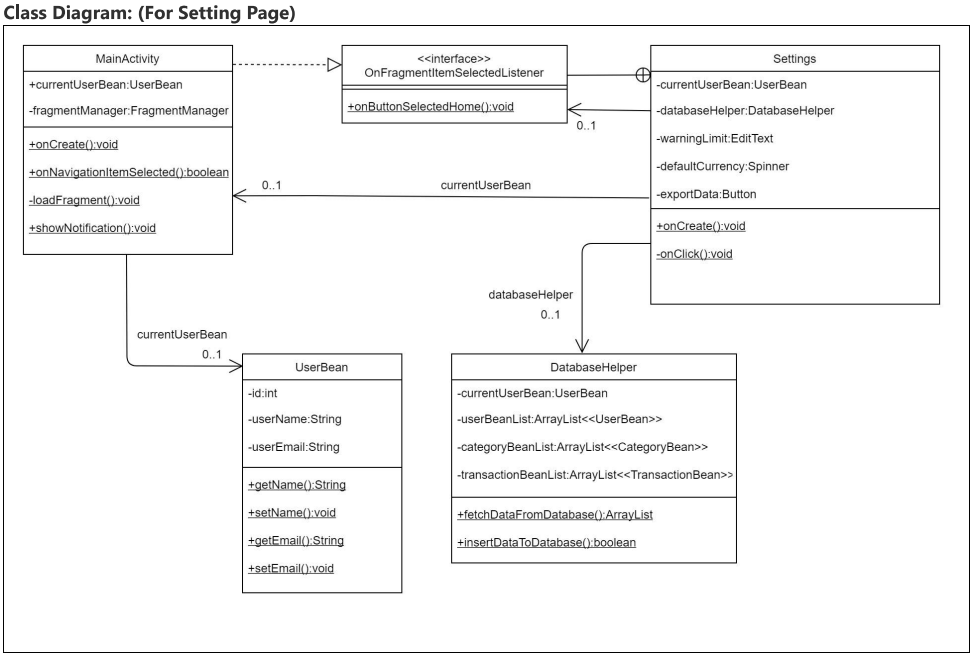
• User can raise an issue in the form of a support contact form. This helps the user to get in touch with the developers for issues which cannot be solved through the help menu feature.

Package Diagram:



• We have added only two functionalities as part of the beta prototype. So the package diagram remains the same with the addition of the raise an issue part.

Class Diagram: (For Setting Page)



• 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 Export functionality is implemented in the settings page.

• Main Activity holds the object of current user (when user signs up or signs in). Settings Fragment has association with Main Activity for calling that current user object. Using this, Settings Fragment communicates with Database Helper with 0..1 multiplicity (because Database Helper single object has enough functionalities).

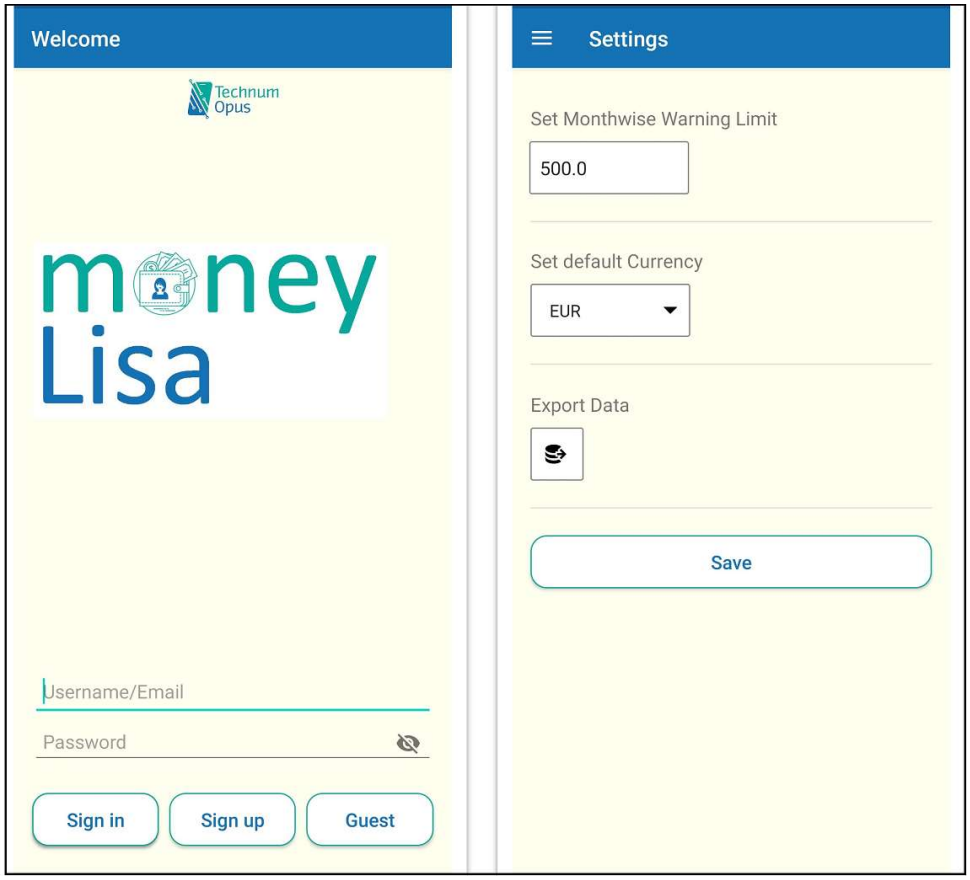
• Main Activity Implements the listener interface which is in composition with settings class

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

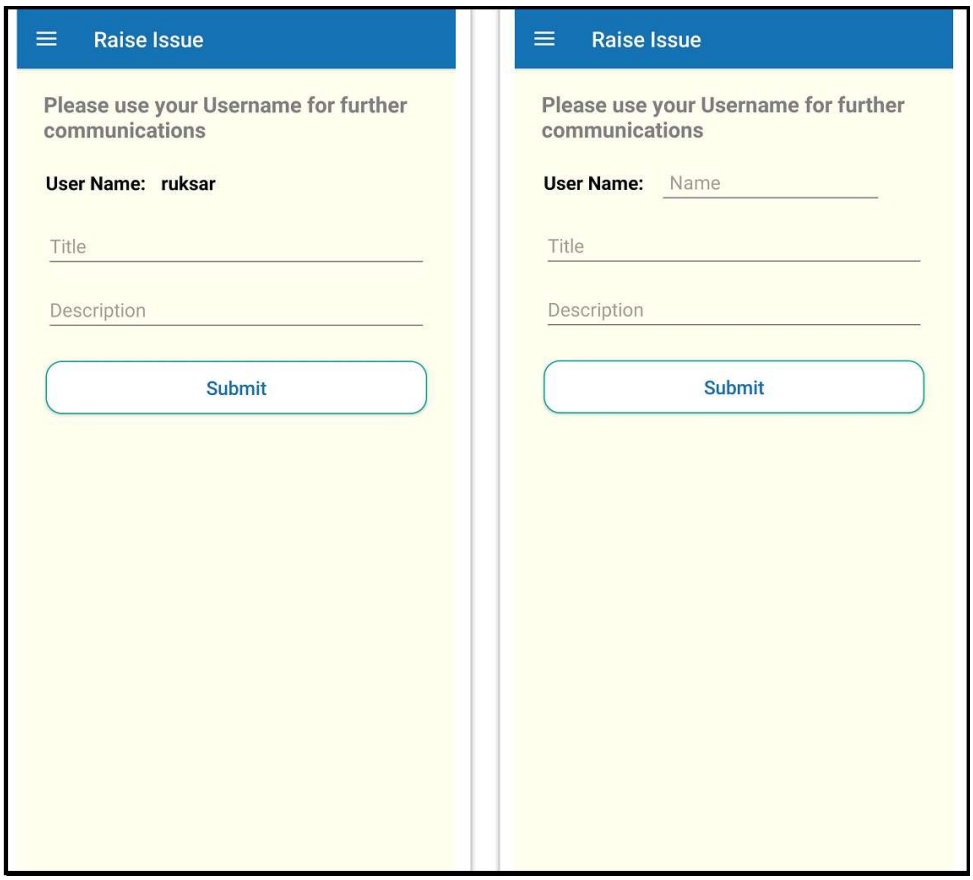
https://code.ovgu.de/steup/technum-opus/-/tree/patch-2/Diagrams/Beta_Prototype

Screenshots and APK

Sign-in with password hide/see & export option



Raise issue for signed in user and guest



APK

moneylisa_v03.apk

This Brings us to the end of our blog. We will see you in our next blog article with our final product. Tschüß!

References:

https://www.guru99.com/dynamic-testing.html

https://www.photocollage.com/

https://app.diagrams.net/

https://online.visual-paradigm.com/