

Final Report for La-Luna

Bilal Al Malek, Ali Alkhaled, Deaa Khankan, Ali Malla

date 2021-10-23

Innehållsförteckning

1	Introduction RAD	4
1.1	Definitions, acronyms, and abbreviations	4
2	Requirements	5
2.1	User Stories	5
2.1.1	Smidig navigation på applikationsnivå	5
2.1.2	Tilläggnig av nya utgifter	5
2.1.3	Lista över tidigare utgifter	5
2.1.4	Kvarstående pengar	6
2.1.5	Tilläggnig av nya kategorier	6
2.1.6	Spendering för aktuell månad	6
2.1.7	Statistisk månadspendering	6
2.1.8	Begränsning av kategoribudgeter	7
2.1.9	Spendering på kategorinivå	7
2.1.10	Kategorisida	7
2.1.11	Redigering av registrerade utgifter	8
2.1.12	Borttagning av registrerade utgifter	8
2.1.13	Hjälp-sidan och rekommendationer	8
2.2	Definition of Done	9
2.3	User interface	9
2.3.1	Färg	9
2.3.2	Övergripande	9
2.3.3	Hemsidan (Home page)	10
2.3.4	Analyssidan (Analysis Page)	13
2.3.5	Kategorisidan (Categories page)	15
2.3.6	Rekommendationer/Hjälp-sidan (QA Page)	18
3	Domain model	18
3.1	Class responsibilities	18
3.1.1	Category	19
3.1.2	Expense	19
3.1.3	IDatabaseHandler	19
3.1.4	SqliteHandler	19
3.1.5	DBHandler	19
3.1.6	View klasser	20
3.1.7	ViewModel klasser	20
4	Introduction SDD	20
4.1	Definitions, acronyms, and abbreviations	20
5	System architecture	21

6	System design	22
6.1	UML package diagram	22
6.2	UML class diagram	23
6.2.1	model package class diagram	23
6.2.2	Home package class diagram	24
6.2.3	Analysis package class diagram	25
6.2.4	Categories package class diagram	26
6.3	Sequence diagram	27
6.4	Design patterns	28
7	Persistent data management	29
8	Quality	29
9	Peer review	30
10	References	31

1 Introduction RAD

La Luna är en budget Applikation som hjälper användaren att ha koll på hur mycket pengar den spenderar varje månad. Idén går på att det ska finnas olika kategorier där användaren lägger en gräns på hur mycket hen tänker spendera i en specifik kategori. När man närmar sig att överstiga gränsen får man en typ av pliancy för att dra uppmärksamhet att användaren behöver spendera mindre i denna kategori för att nå spenderingsmål. Defaulta kategorier ska finnas redan när man kör applikationen för första gången men användaren har möjlighet att lägga till sina egna kategorier för att det ska passa sin spenderingsstil.

Applikationen ger användaren möjlighet att ha koll på sin spending i tidigare månader genom en analys som visas på ett modernt sätt som är enkelt att följa. Applikationen är ganska flexibel då man har möjlighet att modifiera kategorier och utgifter när man råkar mata in felaktiga information. Man kan även filtrera utgifter så att bara utgifterna i en specifik kategori visas.

1.1 Definitions, acronyms, and abbreviations

- Primary key: En kolumn som måste innehålla ett unikt värde på varje rad i en tabell
- Foreign key: En kolumn i en databastabell som pekar mot ett Primary key i en annan tabell
- Activity: Ett fönster som användaren ser i en android applikation.
- Fragment: Ett återanvändbar gränssnitt, man kan skapa den och använda den i olika Activities.
- User Story: En övergripande beskrivning av en funktion i applikationen ur ett användarvänligt perspektiv. Vanligen skriven i ett informellt språk.
- Navigationbar: Ett grafiskt komponent som vanligen placeras i botten av mobilskärmen och innehåller några ikonbeskrivande och/eller textbaserade knappar. Var och en av dessa knappar leder till olika destinationer i applikationen.
- Kategori: En viss gruppering till utgifter som kan skapas av användaren.
- Pliancy: Ett omfattande koncept inom interaktionsdesign och som visar användaren vad hen kan göra i applikationen genom att de olika grafiska komponenten i applikationen bland annat ändrar sina färger om de blir påpekade av musen.
- Visningslista(listview): En vy som grupperar flera identiska element för att sedan visa upp dem i en oftast vertikal skrollbar lista.

2 Requirements

2.1 User Stories

2.1.1 Smidig navigation på applikationsnivå

Description

"Som användare vill jag en navigationbar för att navigera enkelt mellan samtliga sidor (Activities, Fragments, etc)"

Confirmation

Functional:

- Kan jag gå tillbaka till hemsidan?
- Kan jag se på vilken sida jag befinner mig i (Pliancy)?

Non-functional:

- Är det smidigt att navigera mellan applikationens sidor?

2.1.2 Tilläggning av nya utgifter

Description

"Som användare vill jag ha möjlighet att lägga till utgifter själv så att jag har kontroll över min budget samt vilka utgifter som ska ingå i min ekonomiska planering på applikationen."

Confirmation

Functional:

- Kan jag lägga till en utgift när som helst?
- Kan jag välja vilken kategori denna utgift kommer att tillhöra?

Non-functional:

- Kan jag se alla mina utgifter när som helst?

2.1.3 Lista över tidigare utgifter

Description

"Som användare vill jag kunna ha en lista med tidigare registrerade utgifter så att jag kan veta var jag har spenderat mina pengar."

Confirmation

Functional:

- Kan jag öppna en viss registrerad utgift för att få detaljerad information om den?
- Kan jag redigera denna list?
- Kan jag radera en tidigare registrerad utgift från denna lista?

Non-functional:

- Innehåller denna lista endast utgifter till den aktuella månaden?
- Hur ser denna lista ut när den är tom?

2.1.4 Kvarstående pengar

Description

"Som användare vill jag veta hur mycket pengar som finns kvar av budgeten så att jag kan se hur mycket jag har spenderat i helhet."

Confirmation

Non-functional:

-Står de kvarstående pengarna tydligt i samtliga sidor där jag förväntas vilja ta reda på dem?

2.1.5 Tilläggning av nya kategorier

Description

"Som användare vill jag kunna lägga till en ny kategori ifall det inte finns en default kategori som överensstämmer med det jag har köpt."

Confirmation

Functional: -Kan jag lägga till en ny kategori?

-Kan jag välja en budget till den nya kategorin?

Non-functional:

-Kan jag när som helst lägga till en ny kategori?

-Finns det en gräns till antal kategorier som jag kan ha/lägga till?

2.1.6 Spendering för aktuell månad

Description

"Som användare vill jag kunna se hur mycket jag har spenderat inom varje kategori under månaden så att jag kan ha koll på hur min spendering går till."

Confirmation

Functional:

-Kan jag se hur mycket jag har spenderat inom varje kategori (månadsvis)?

-Kan jag veta hur mycket budget som är kvar för varje kategori?

Non-functional:

-Kan jag se hur mycket budget som är kvar för varje kategori när som helst ?

2.1.7 Statistisk månadspendering

Description

"Jag vill kunna se hur mycket jag har spenderat inom varje kategori från tidigare månader så att jag som användare ska ha koll på hur mina utgifter varjerar under året"

Confirmation

Functional: -Kan jag klicka på Analys i navigationsbar?

-Kan jag se en lista med alla kategorier i form av en grid (Grid of Equals)?

-Kan jag se budgetten på varje kategori?

-Kan jag se hur mycket jag har spenderat inom varje kategori (årligen)?

2.1.8 Begränsning av kategoribudgeter

Description

"Som användare vill jag kunna begränsa budgeten av varje kategori för att det ska bli lättare att kontrollera hur mycket pengar jag tänker att spendera på respektive kategori."

Confirmation

Functional:

-Kan jag välja vilket belopp som helst?

Non-functional:

-Är det möjligt att ändra på budgeten när som helst?

2.1.9 Spendering på kategorinivå

Description

"Som användare vill jag kunna se en lista med de senaste registrerade utgifter som tillhör en viss kategori så att jag kan lätt komma till mina utgifter inom den"

Confirmation

Functional:

-Får jag en ny sida som visar en lista med utgifter om jag klickar på en kategori?

-Kan jag gå tillbaka till alla kategorier?

Non-functional:

-Kan jag öppna de senaste utgifter inom en kategori när som helst?

2.1.10 Kategorisida

Description

"Som användare vill jag ha en sida till kategorier så att jag kan modifiera dem på ett smidigt sätt."

Confirmation

Functional:

-Kan jag klicka på kategorier i navigationsbar?

-Kan jag se en lista med samtliga kategorier när jag klickar i navigationsbaren?

-Kan jag se relevant information om varje kategori (budget, namn, osv.....)

-Kan jag klicka på "Edit" och ändra kategorins information?

2.1.11 Redigering av registrerade utgifter

Description

"Som användare vill jag kunna redigera en utgifts information ifall jag har råkat skriva felaktig information t.ex. (pengar, kategori, etc)"

Confirmation

Functional:

- Kan jag klicka på en utgift och få en sida med ändringar?
- Kan jag klicka på spara ändringar och se att ändringarna sparades?
- Kan jag klicka avbryt och gå tillbaka utan att ändra?

Non-functional:

- Funkar det närsomhelst så länge jag har utgifter?
- Kan jag gå tillbaka utan att ändra i fall jag klickade fel?

2.1.12 Borttagning av registrerade utgifter

Description

"Som användare vill jag kunna ta bort en registrerad utgift för att det finns en risk att jag bland annat råkar skriva in felaktig information."

Confirmation

Functional:

- Kan jag radera en utgift?

Non-functional:

- Kan jag ta bort tidigare tillagda utgifter när som helst?

2.1.13 Hjälp-sidan och rekommendationer

OBS! Ej implementerad användarupplevelse

Description

"Som användare vill jag erbjudas några rekommendationer om hur man kan spara pengar för att jag som oerfaren ska kunna spara pengar och planera inför nästa månad."

Confirmation

Functional:

- Kan jag se alla rekommendationer?
- Är texternas stil tydlig?
- Är rekommendationerna hjälpsamma?

2.2 Definition of Done

För att en användarberättelse, User story, ska definieras som fullständigt kompletterad ska följande krav ha uppfyllts:

- *Kompletterade uppdrag*
Samtliga uppdrag inom en användarberättelse ska vara färdigimplementerade.
- *Exekverbar kod*
Det måste finnas en fungerande kod som kompileras och exekveras utan några tekniska problem.
- *Testning*
Testning inledas med tillräckligt många visuella exempel för att sedan en regelbunden testning ska påbörjas, så som enhetstest (Unit test). Denna fas är väldigt viktig inom mjukvaruutveckling eftersom den bekräftar att koden beter sig som förväntat, vilket i sin tur leder till hög mjukvarukvalitet.

2.3 User interface

2.3.1 Färg

Applikationen har en enhetlig visuell struktur där alla sidorna inte skiljer sig mycket från varandra. Färgerna som används i applikationen är färger som gör applikationens utseende något sportig, vilket ger applikationen en vacker utsikt och större önskan att användas av yngre målgrupp. De mest använda färgerna i applikationen börjar från vänster på färgpaletten vilket åskådliggörs i Figur 1.

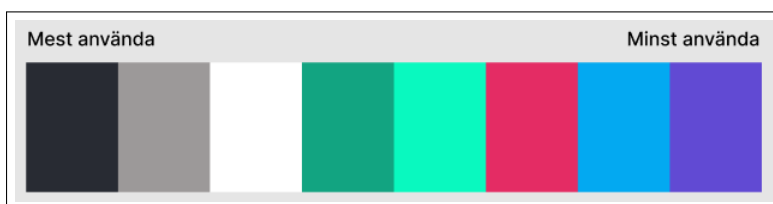


Figure 1: Färgerna som används genomgående i applikationen.

2.3.2 Övergripande

Applikationen består av fyra huvudsidor som användaren kan navigera mellan dem när som helst med hjälp av den globala menyn längst ner på skärmen, se

figur 2.

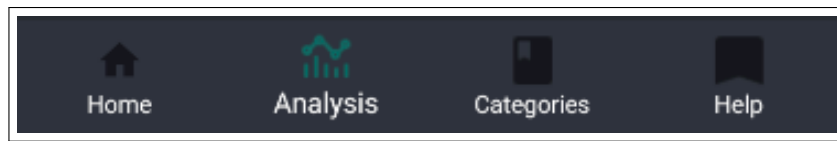


Figure 2: Den globala menyn.

Denna globala meny innehåller en pliancy som reagerar med användarens reaktion. Till exempel, om användaren klickar på Analysis-sidan, ändras färgen på analysis-ikonen i den globala menyn. Därtill är interaktiva komponenter förtydliga med hjälp av ikoner. Denna meny är designat för att göra navigeringen mellan sidorna enklare och applikationen mer överskådligt.

2.3.3 Hemsidan (Home page)

Hemsidan är den första sidan som dyker upp när användaren öppnar applikationen varje gång. Den består av ett cirkeldiagram som visar användaren hur mycket av budgeten som har spenderats och justerar dess färg efter förbrukade pengar. Dessutom finns det en textview längst upp till vänster på skärmen vilka visar användaren hela budgeten, se figur 3.

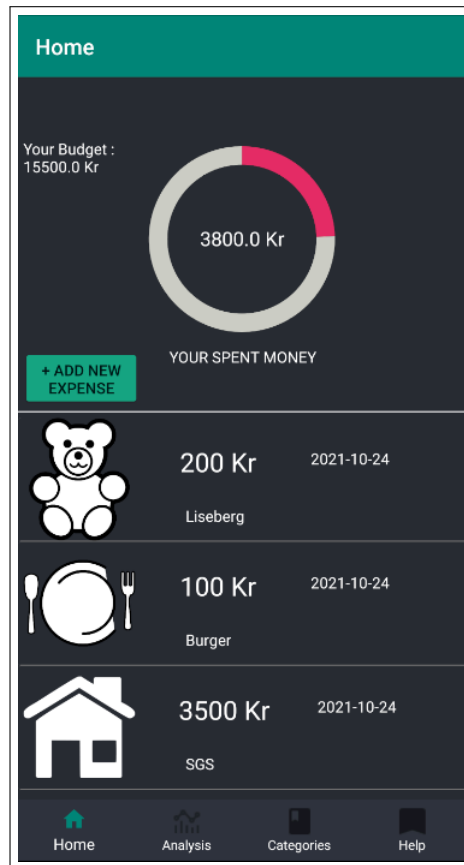


Figure 3: Hemsidan.

I den nedre halvan av gränssnittet ligger en rullningsbar visningslista (listview) som visar användaren alla utgifter som har lagts till. Utgifterna placeras kronologiskt efter datum i visningslistan, dvs. senast först. Därtill finns det också en grön knapp så att användaren kan lägga till sina utgifter. Genom att trycka på knappen visas en ny sida där användaren måste ange information om den utgiften som ska läggas till, vilket åskådliggörs i Figur 4.

The screenshot shows a mobile application interface with a teal header bar containing the text "La luna". Below the header, the background is a light gray. The form consists of three input fields: "Expense name:" with a text input, "Expense value:" with a text input, and "Expense category:" with a dropdown menu currently showing "Food". At the bottom of the form is a teal button labeled "SAVE".

Figure 4: Lägga till en utgift sidan.

Dessutom har användaren också möjlighet att redigera en utgift genom att klicka på utgiften i visningslistan. Detta öppnar ett nytt gränssnitt med detaljerad information om utgiften som går att redigera, se figur 5.

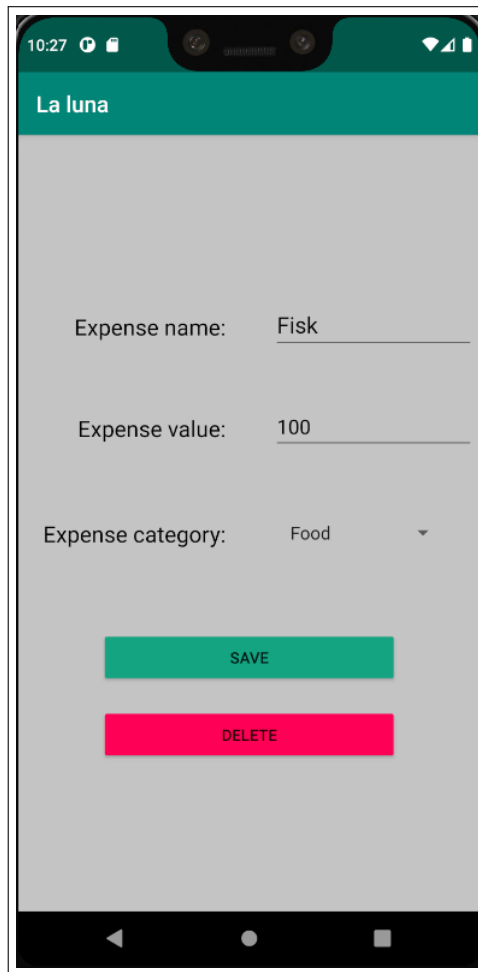


Figure 5: En sida där användaren kan läsa och redigera en utgiftsinformation samt radera den.

2.3.4 Analyssidan (Analysis Page)

I den övre halvan av gränssnittet finns samma komponenter som i den övre halvan av hemsidan. Ett litet tillägg som finns i analyssidan är två små pilar för att ta användaren till föregående analyssidor. Utöver detta finns det små cirkeldiagram som liknar det stora cirkeldiagram i toppen, men istället för att visa användaren hur mycket av hela budgeten som har spenderats, visas bara hur mycket som har spenderats av budgeten för varje kategori, se figur 6.

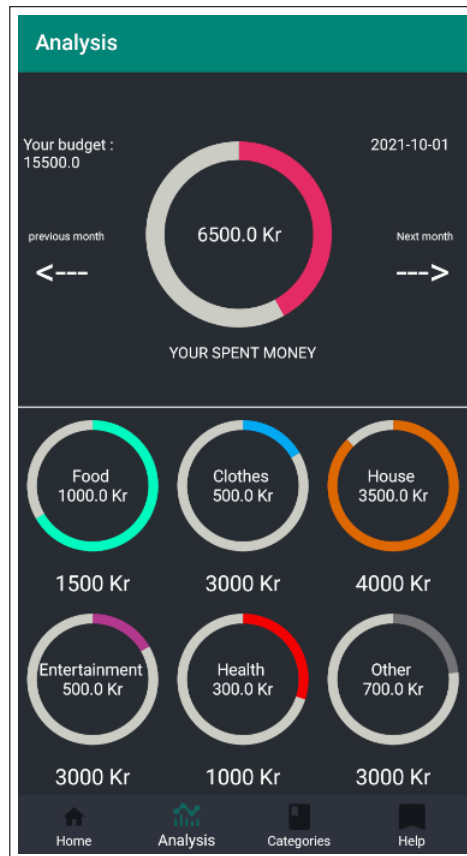


Figure 6: Analyssidan.

Genom att klicka på ett cirkeldiagram öppnas ett nytt gränssnitt där visas en lista över specialutgifter som tidigare lagts till för den valda kategorin. Till exempel, om användaren bara vill se utgifterna för Matkategorin "Food Category", kan hen klicka i den lilla gröna cirkeln, för klargörande se figur 7.

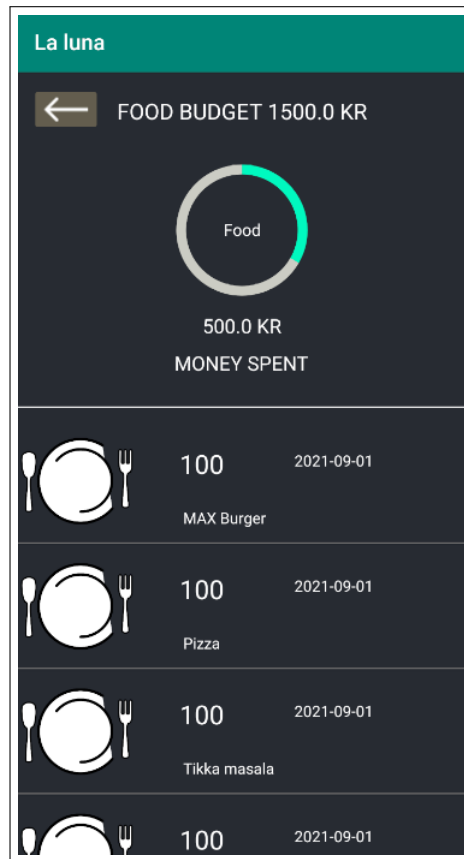


Figure 7: En lista av utgifter som tillhör till Mat kategori.

2.3.5 Kategorisidan (Categories page)

Kategorisidan innehåller en lista av kategorier som nästan täcker hela gränssnittet i form av en rullningsbar visningslista (listView). Dessa kategorier är standardkategorier och kategorier som användaren tidigare har lagt till. Dessutom finns det en knapp längst upp till vänster för att lägga till kategorier vilket åskådliggörs i Figur 8.

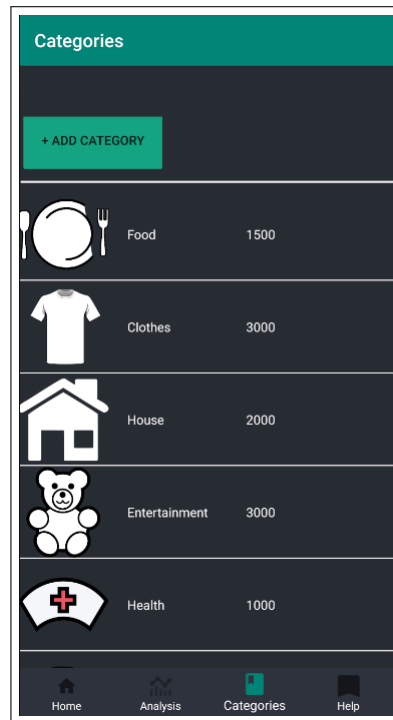


Figure 8: Kategorisidan.

Processen för att lägga till en ny kategori eller läsa kategorisinformation, fungerar på liknande sätt som för utgifter vilka ligger under rubriken hemsidan. Men för mer tydlighet se figur 9 och 10.

La luna

ADD A NEW CATEGORY

Name:

Budget:


ADD CANCEL

This is a mobile app interface for adding a new category. It features a teal header with the text 'La luna'. Below the header, the title 'ADD A NEW CATEGORY' is centered. There are two input fields: 'Name:' and 'Budget:', each followed by a text input line. At the bottom, there are two buttons: a teal 'ADD' button and a red 'CANCEL' button.

Figure 9: Processen för att lägga till en ny kategori.

La luna

CATEGORY INFORMATION



Name: Food

Budget: 1500

SAVE CANCEL

DELETE

This is a mobile app interface for viewing and editing category information. It features a teal header with the text 'La luna'. Below the header, the title 'CATEGORY INFORMATION' is centered. To the right of the title is a small black square icon with a white pencil. There are two input fields: 'Name:' with the text 'Food' and 'Budget:' with the text '1500'. At the bottom, there are three buttons: a teal 'SAVE' button, a red 'CANCEL' button, and a red 'DELETE' button.

Figure 10: processen för att läsa information om en specifik kategori samt radera eller redigera en kategori.

För att kunna redigera måste användaren först klicka på pennikonen sedan redigera. För att spara ändringarna behöver användaren klicka på knappen "spara" (Save button). En kategori raderas genom att användaren klickar på knappen "Ta bort" (Delete button)

2.3.6 Rekommendationer/Hjälp-sidan (QA Page)

Den här sidan är tom. Vi strävade efter att göra det till en hjälpsida för användaren som innehåller en lista med frågor och svar, men vi gjorde det inte eftersom vi inte hade tillräckligt med tid.

3 Domain model

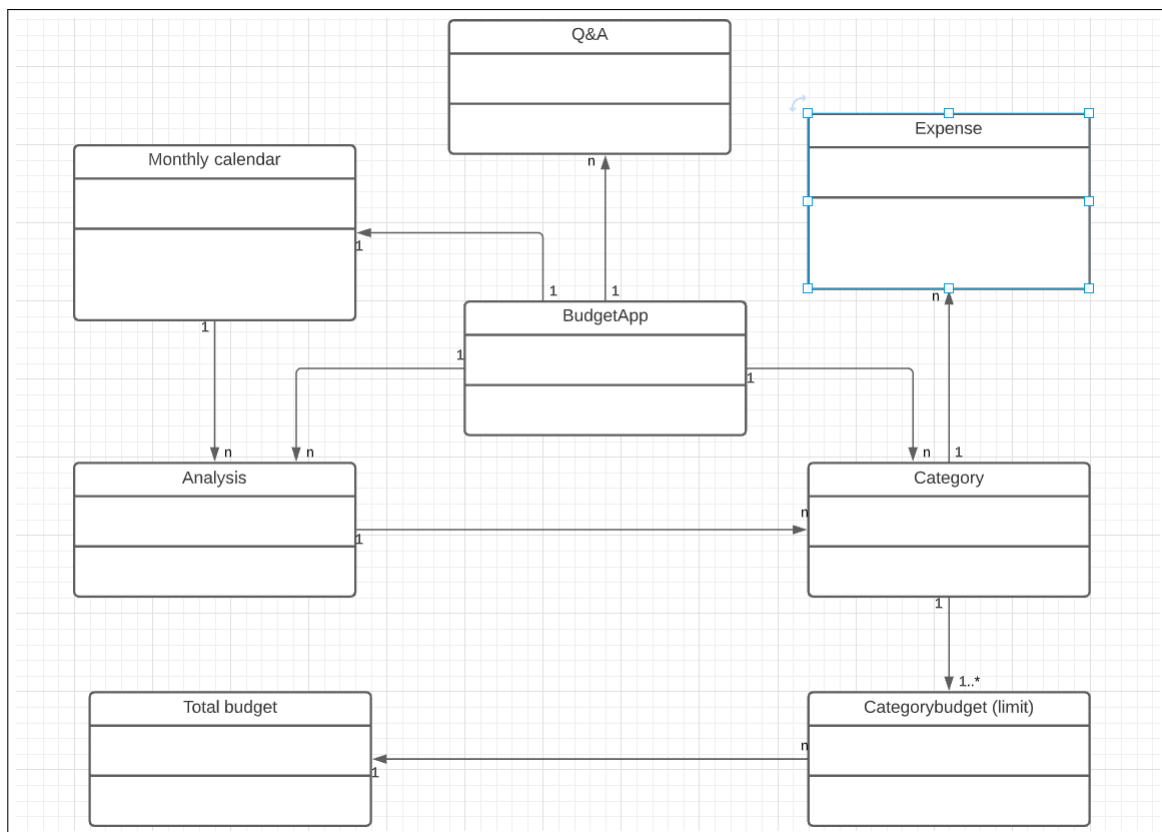


Figure 11: Domain model.

3.1 Class responsibilities

Här är en lista med de klasser som ska användas i applikationen och deras ansvar.

3.1.1 Category

Klassen innehåller information om en kategori i databasen i form av attributes till exempel (id, namn, gräns, startdatum, slutdatum). Varje objekt av Category klassen ska representera en rad i tabllen Categories i databasen och den ska användas av applikationens gränssnitt.

3.1.2 Expense

Klassen innehåller information om en utgift, samma som kategori klassen den har utgifts info i form av attributes (namn, värde, osv) och varje objekt av den ska representera en rad i tabellen Expenses. Databastabellen som klassen representerar innehåller en (Foreign key) så klassen innehåller en attribute av typen Category som pekar mot ett kategori. Den kommer att användas i applikationens gränssnitt också.

3.1.3 IDatabaseHandler

En interface, varje klass som implementerar av denna interface ska vara ansvarig för hantering av databasen alltså den ska innehålla metoder som är ansvariga för hämtning och inmatning av information från databasen.

3.1.4 SqliteHandler

Klassen är ansvarig för hanteringen av sqlite databasen som ska finnas i mobilen. Den implementerar interfacet IDatabaseHandler och innehåller metoder för hämtning och inmatning av information från databasen. Exempelvis på dessa metoder är getCategories som ger tillbaka alla kategorier i databasen samt addExpense som lägger till en utgift i databasen. Klassen är också ansvarig för att skapa databasen och skapa tabellerna som finns i den eftersom den ärver från abstrakt klassen SQLiteOpenHelper som är uppbyggd inom android. Databasen skapas i konstruktorn och tabellerna skapas i metoden onCreate.

3.1.5 DBHandler

Klassen är ansvarig för kommunikationen mellan databasen och resten av programmet. Den innehåller ett objekt som har den dynamiska datatypen SqliteHandler och den statiska data typen IDatabaseHandler. Den använder sig av denna objekt för hanteringen av databasen och den underlättar för bytet av databasen också och gör att resten av programmet är oberoende av vilken databas som används i bakgrunden.

3.1.6 View klasser

Design mönstret MVVM har används för att bygga applikationen. Det betyder att varje Activity och Fragment som finns i applikationen innehåller en Java klass som är ansvarig för att hantera gränssnittet. Dessa är View klasser.

3.1.7 ViewModel klasser

Det behövs viewmodel klasser i mvvm design pattern som kommer att fungera som en länk mellan model klasser (Category, expense, DBHandler) och view klasser. Viewmodel klasser kommer att ärva från BaseObservable klass som är uppbyggd inom android och fungerar som hjälpmedel för data utbyte mellan view klasser och viewmodel klasser

4 Introduction SDD

La Luna är en budget Applikation som hjälper användaren att ha koll på hur mycket pengar den spenderar varje månad. Applikationen är uppbyggt med syfte att den ska följa objekt orienterade principer. Exempelvis på dessa principer är MVVM som används för att separera logik från gränssnitt samt minska kopplingar mellan klasserna. Facade Pattern som döljer databasen från klientkoden samt Bridge Pattern som förenklar bytet av databasen och gör det möjligt att modellen blir oberoende av services (SQLite) vilket är ett krav på applikationen. Applikationen använder sig av SQLite som databas för att spara data som matas in av användaren. Applikationens modell har testats med JUnit samt med hjälp av PMD analys plug-in.

4.1 Definitions, acronyms, and abbreviations

- MVVM: Ett designmönster som strukturerar upp mjukvaruprojekt genom att uppnå det så kallat Separation of Concern, vilket med andra ord innebär att gränssnitt, affärslogik och data ska skiljas och vara direkt oberoende av varandra.
- SOLID: En förkortning för fem viktiga designprinciper inom mjukvaruutveckling, vilka är Single responsibility principle, Open closed principle, Liskov substitution principle, Interface segregation principle och Dependency inversion principle.
- Databas: En samling information/data sparad i ett datorsystem.
- SQLite: Ett tillgängligt bibliotek för server-lös databashanterare.

- DBHandler: En klass som underlättar kommunikationen mellan databasen och andra delar i projektet

5 System architecture

La Luna är en Android applikation som är skriven i Javaspråket. Applikationen följer MVVM desigmönstret vilket är viktig för att strukturera applikationen på ett sätt som följer SOLID principer och ha en low coupling kod. Applikationen använder sig av SQLite för att spara de data som matas in av användare såsom utgifter och kategorier. Applikationens modell är oberoende av SQLite databas då kommunikationen sker genom en medellager klass "DBhandler" som hanterar mellan modellen och service "SQLite". Genom detta kan man enkelt byta databasen vilket är viktig för att koden ska vara öppen för förlängning och stängd för modifiering.

När man kör applikationen, öppnas hem-sidan först. Där går man oftast för att se tidigare utgifter och lägga till nya eller modifiera redan existerade utgifter. Om användaren till exempel vill lägga till en utgift, behöver hen klicka på knappen "Add Expense" sedan behöver hen fylla utgiftsinformation. Därefter sparas utgiften i tabellen (SQLite databasen) med namn, värde, skapelsedatum och till vilken kategori den tillhör plus ID nummer. För att se en övergripande bild på utgifterna som är delade i olika kategorier går man till analys-sidan. Där också visualiseras Kategorierna genom cirkeldiagram (pie-charts). Dessutom kan man vara i behov att lägga till nya kategorier vilket man gör i kategori sidan och datan sparas på liknande sätt som när sparas en utgiftsdata. När applikationen stängs och öppnas igen visas all data som lagts innan.

6 System design

6.1 UML package diagram

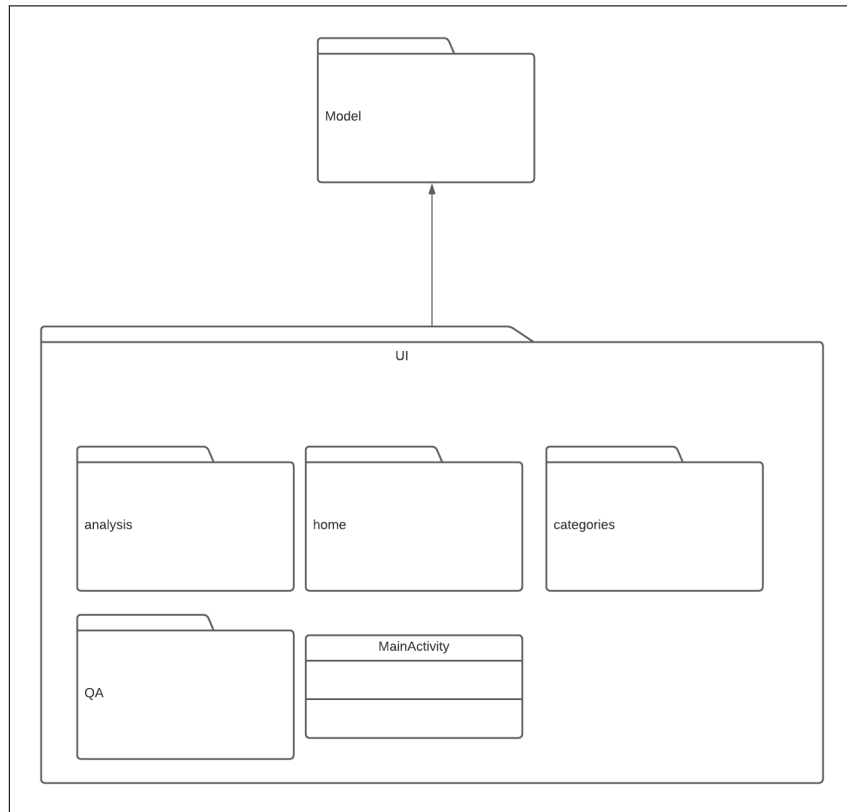


Figure 12: UML package diagram.

6.2 UML class diagram

6.2.1 model package class diagram

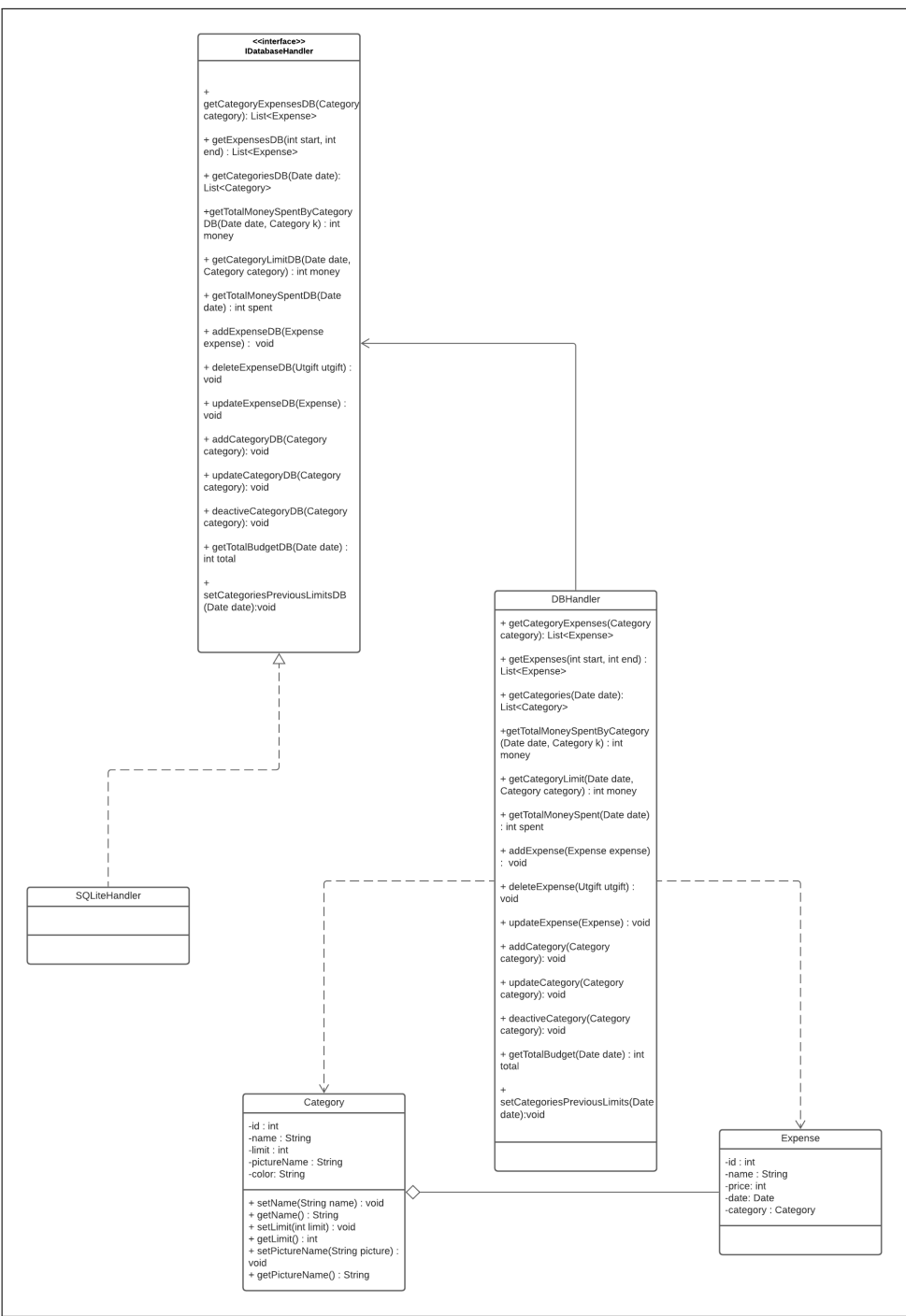


Figure 13: Model class diagram.

6.2.2 Home package class diagram

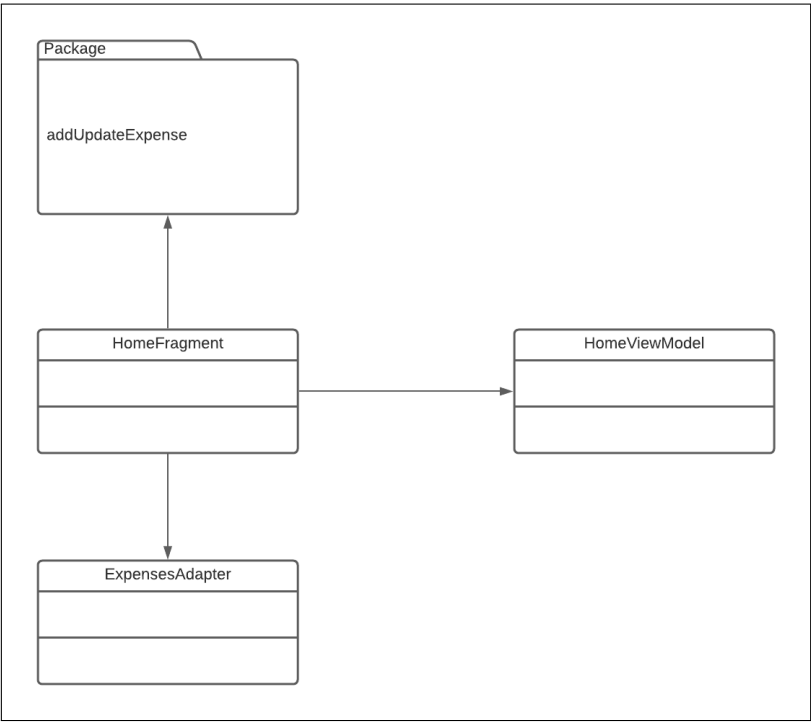


Figure 14: Home class diagram.

addUpdateExpense package class diagram

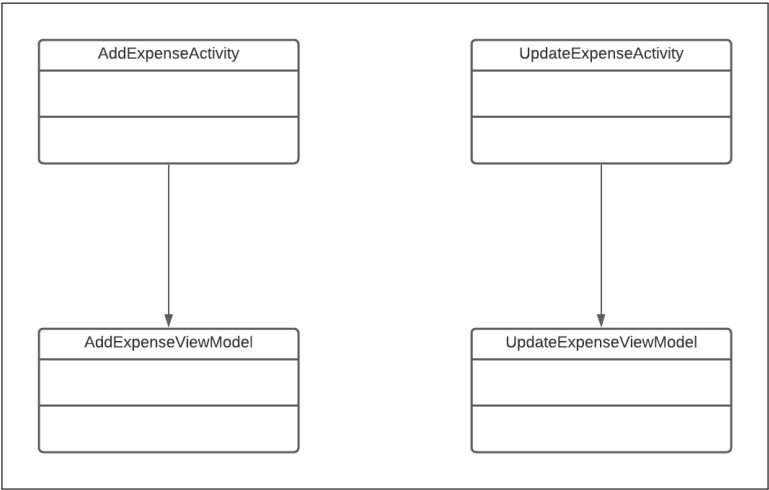


Figure 15: Add update class diagram.

6.2.3 Analysis package class diagram

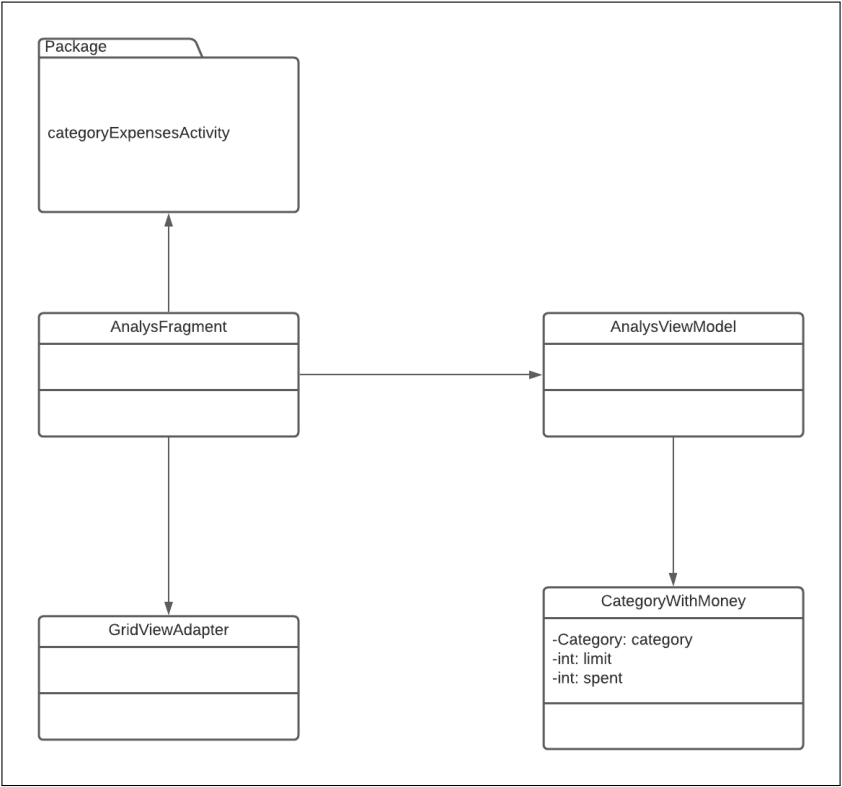


Figure 16: Analysis class diagram.

categoryExpensesActivity package class diagram

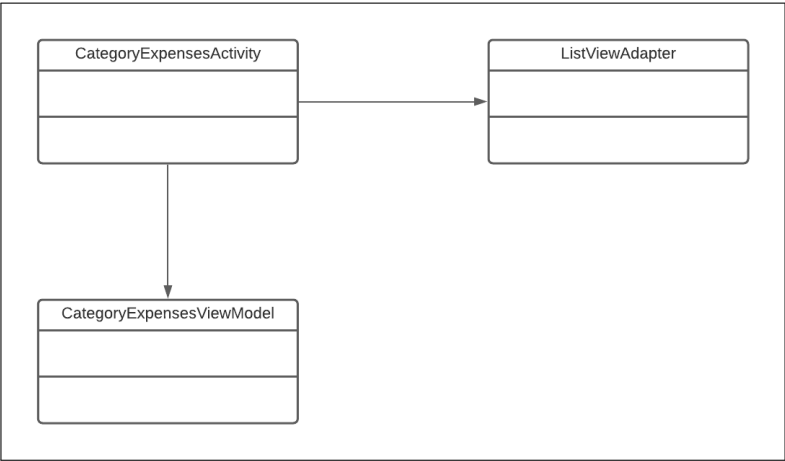


Figure 17: Category expenses class diagram.

6.2.4 Categories package class diagram

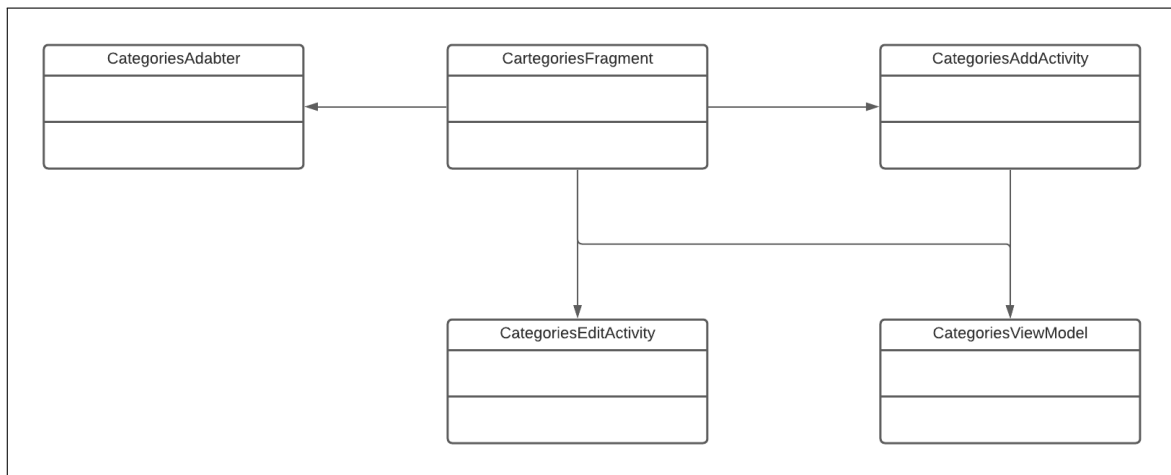


Figure 18: Categories class diagram.

6.3 Sequence diagram

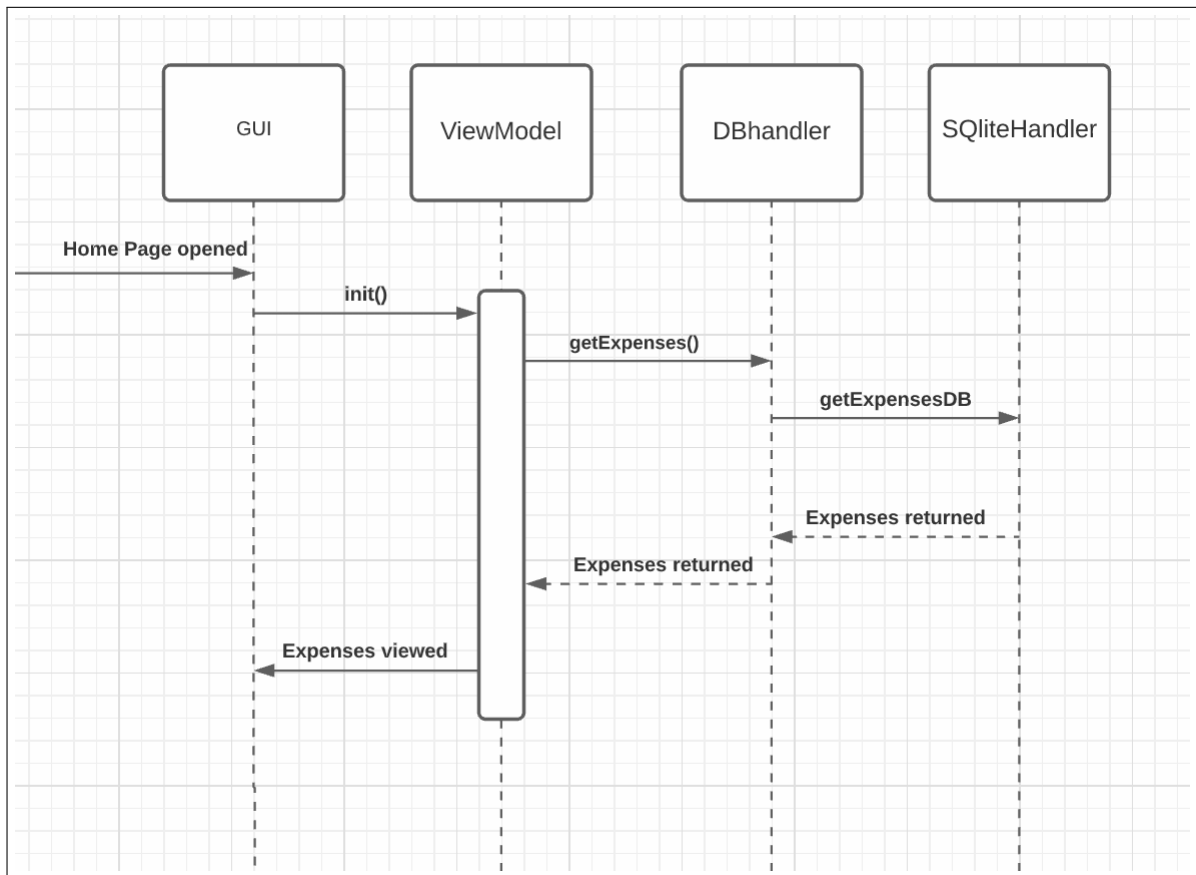


Figure 19: Home Sequence diagram.

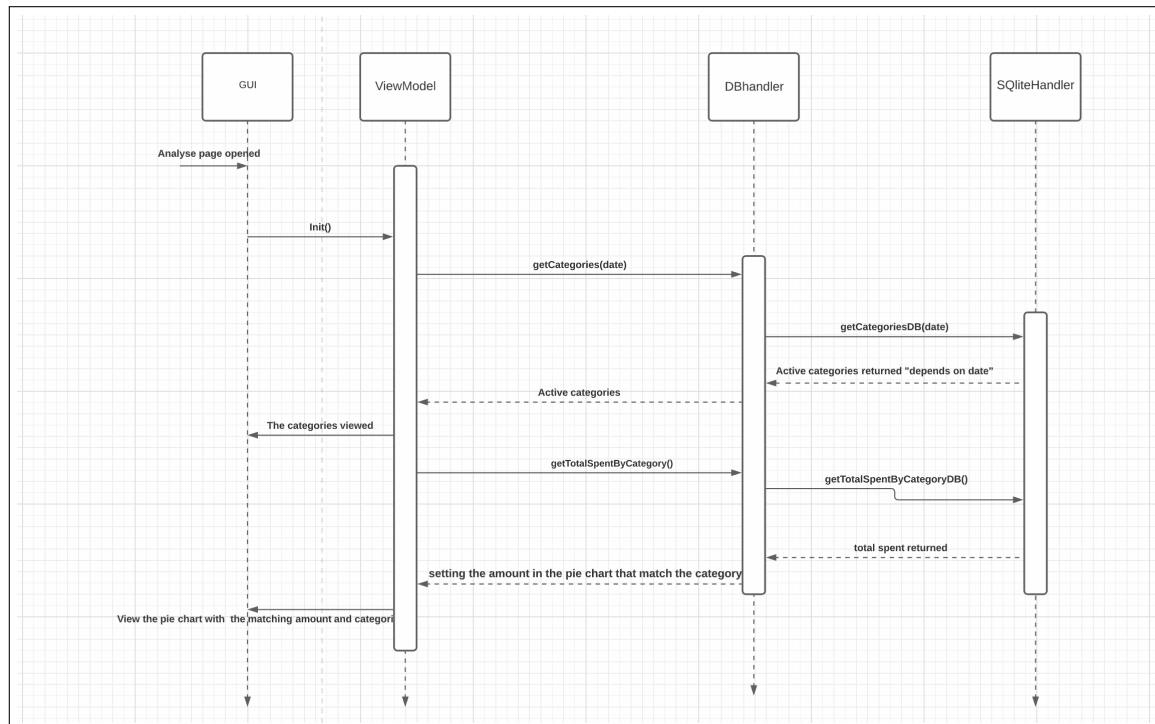


Figure 20: Analysis Sequence diagram.

6.4 Design patterns

- Model-View-ViewModel pattern: Applikationen följer MVVM designmönster. Databasen är inte direkt kopplat till klient koden. Klassen "DBhandler" fungerar som ett lager mellan databasen och klientkoden. Men vi vill inte heller att DBhandler ska ha kopplingar med View klasser, dvs klasser som har renderingsansvar. Därför kommer ViewModel lager för att hämta data från databasen genom DBhandler och skicka den vidare till View. Dessutom kommer det att lyssna på de ändringar som sker i UI för att uppdatera motsvarande data i databasen. Varje View-klass kommer att ha sin egen ViewModel klass som har ovannämnda ansvar.
- Bridge pattern: Bridge designmönster används för att förenkla bytet av databasen. Vi har ett interface som har alla metoder som vår DBHandler behöver. SQLiteHandler klassen ärvar detta gränssnitt. DBhandler beror inte på SQLiteHandler där samtliga SQLite-instruktioner finns, men har istället ett attribut av IDatabasHandler (beror på abstrktion) som pikar i sin tur på ett SQLite-objekt vid runtime. Detta ger oss möjlighet att byta databasen när som helst då det bara är att skapa en ny databasklass och byta referensen som attributen pikar på till den nya databasen.

7 Persistent data management

Det finns data som inte ändras i applikationen. Applikationen kommer att skapa defaulta kategorier som användaren kommer inte att kunna ta bort eller ändra, den enda som användaren kommer att kunna ändra på defaulta kategorier är budgeten. De ska finnas som vanliga kategorier i kategorier tabellen men applikationen kommer att hardkodas så att den vet att tillexempel första fem kategorier ska inte ändras och viewn kommer inte att låta användaren ändra de.

Applikationen kommer att använda bilder också som inte ändras. Det ska finnas ett stort antal bilder som finns på resources mappen, de ska användas för kategorier alltså varje kategori ska ha en bild. Användaren ska se de när hen skapar en ny kategori där hen ska kunna välja en bild för den nya kategorin och varje kategori ska innehålla ett referens nummer i databasen som pekar på en specifik bild.

8 Quality

Test till DBHandler är gjort men vi kan inte se coverage procent av någon anledning. Detta ska lösas i kommande iterationer. Testerna är gjorda med Junit och de finns i klassen DatabaseClassTest. Samtliga metoder i DBhandler har testats på olika sätt för att täcka samtliga möjliga fall och därmed se till att dessa metoder beter sig som förväntat. PMD analys plug-in har använts för att undersöka om koden följer Java standard principer. Bild på resultat finns i referens, Figure 1.

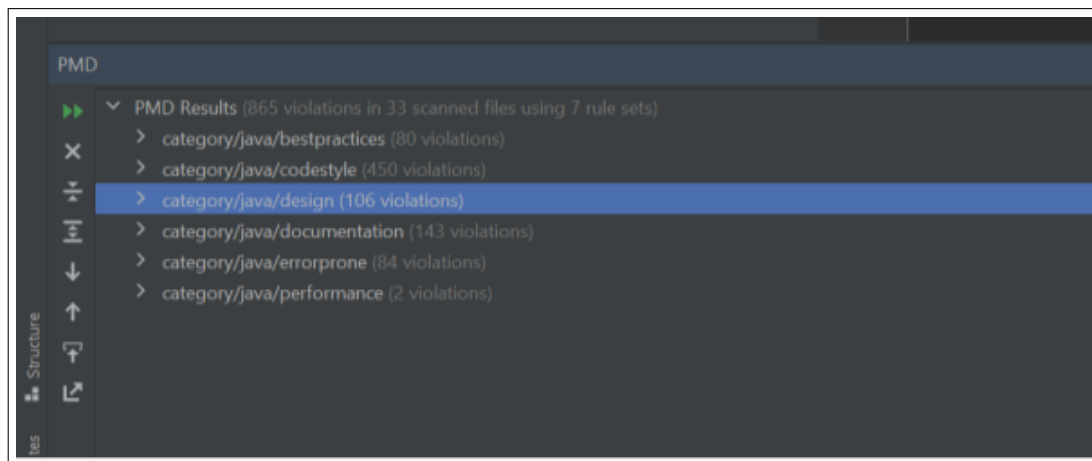


Figure 21: PMD.

..

9 Peer review

Inledningsvis följer kodbasen många principer för objektorienterad design vilket gör koden extensibel och clean till en viss nivå. Eftersom projektet saknade fullständiga RAD och SDD filer var det extra uppmanande att förstå spelet helt. Viss konstruktiv återkoppling som kan förbättra koden har dock tagits fram. Angående designprinciper (speciellt sett SOLID) följer de flesta klasserna Single responsibility principle, men vissa klasser/metoder följer inte den på ett korrekt sätt. Board.java har exempelvis mer än ett ansvar vilket också kan märkas ut genom att läsa dokumentationen som beskriver vad klassen gör. Vissa metoder innehåller ett stort antal av bland annat if-satser (helt normalt då spelalgoritmen förmodligen kräver detta). Dessa metoder skulle möjligtvis kunna delas in till mindre metoder (funktionell nedbrytning) för att göra det enklare att läsa, testa och felsöka koden. Att nästan samtliga klasser är beroende av abstraktion (med andra ord, följer Dependency inversion principle) så resulterar det i att koden också följer Open closed principle. Användning av abstraktion i kodbasen har även bidragit till att undvika mycket duplicerad kod vilket också är bra med kodbasen.

När det gäller designmönster så använder sig kodbasen av vissa designmönster såsom Observer, Factory, Singleton och MVVM. Designmönstret Factory är vanligen användbar när det finns en komplicerad process för att skapa ett visst objekt, såsom att skapa många andra objekt för att sedan kunna skapa det önskade objektet. I sådana fall skulle det resultera i att klientklassen skulle ha onödiga beroenden och då löser Factory mönster detta problem. Som det ser ut i kodbasen nu har många kopplingar, som skulle ha uppstått, undvikts med hjälp av detta designmönster.

Applikationen använder sig av Observer designmönster för att uppdatera sidorna och skicka data från modellen till viewn. Den använder ett interface som heter Isubscriber som implementeras av View-klasserna och modellen har en lista med objekt av klasser som implementerar Isubscriber. Vid eventuella uppdateringar skickar modellen de ändringar som har skett till samtliga subklasser i listan. Problematiken med det är att modellen tvingas att kommunicera direkt med viewn, vilket enligt MVVM inte är tillåtet. All kommunikation mellan dessa moduler ska istället ske indirekt via View-Model. En rimlig lösning till detta problem skulle vara att ViewModel klasserna implementerar interfacet Isubscriber istället och tar emot data från modellen, och därefter skicka de vidare till viewn.

För att denna lösning ska vara lätt implementerad och enkel kan data överföras mellan ViewModel genom att använda LiveData som är ett väldigt bra hjälpmedel i Android. Det möjliggör att viewn lyssnar (Observe) på vissa MutableLiveData objekt som ska ha skapats i ViewModel och uppdaterar data så fort något av dessa objekt har ändrat sin status.

Koden är även väldokumenterad men en sak som skulle kunna förbättras kan vara att använda ett enda språk vid dokumentationen och inte blanda både svenska och engelska. Dessutom finns det lagom många tester. Det var lite svårt att avgöra om kodbasen är tillräckligt testad då det inte gick att köra tester med Coverage-funktionen.

Övriga generella förbättringar kan vara att bli av med onödig kod. Exempelvis finns det inget behov av metoden `indexOfPoint()` eftersom det finns redan en identisk metod i Java som beter sig exakt samma, `indexOf()` i `ArrayList`. Samma sak gäller metoden `isInList()` som kan ersättas av den färdiga metoden `contains()`

10 References

Tidwell, Jenifer, et al. Designing Interfaces : Patterns for Effective Interaction Design, O'Reilly Media, Incorporated, 2020. ProQuest Ebook Central, <https://refactoring.guru/design-patterns>
<https://codingwithmitch.com/blog/getting-started-with-mvvm-android/>