System design document for La-Luna

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

date 2021-10-10
version
1

Innehållsförteckning

1 Introduction

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 varje kategori. När man närmar sig att överstiga gränsen får man en typ av pilancy 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 möjligheten att ha koll på sin spendering i tidigare månader genom en analys som vissas på ett modernt sätt som är enkelt att följa. Applikationen är ganska flexible 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 vissas. Det ska finnas en ”sida” i applikationen där man kan få några rekommendationer för att spendera mindre.
Applikationen är uppbyggt med syfte att den följer MVVM patern. Applikationens modell ska vara oberoende av services samt den ska följa open closed princip.

1.1 Definitions, acronyms, and abbreviations

- MVVM: Ett designmönster som strukturerar upp mjukvaruprojekt genom att uppnå det så kallat Seperation of Concern, vilket med andra ord innebär att gränsnitt, 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 tillgänligt bibliotek för server-lös databashanterare.
- DBHandler: En klass som underlättar kommunikationen mellan databasen och andra delar i projektet

2 System architecture

La Luna är en Android applikation som är skriven i Javaspråket. Applikationen följer MVVM designpatern 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 mellan modellen och service ”SQLite”. Genom detta kan man enkelt byta databasen vilket är vikting för att koden ska vara open för extension och closed för modifation. När man kör applikationen så är den i Home sidan. Där går man oftast för att se vilka utgifter den har samt lägga till nya eller modifiera redan existerade utgifter. Man går till Analys sidan för att se en övergripande bild på utgifterna som är delade i olika kategorier. Kategorierna visualiseras genom pie-chart diagram. Man kan vara i behöv att lägga till nya kategorier vilket man gör i kategori sidan. När man är klar och stänger av applikationen då sparas allt i SQLite databasen.

3 System design

3.1 UML package diagram

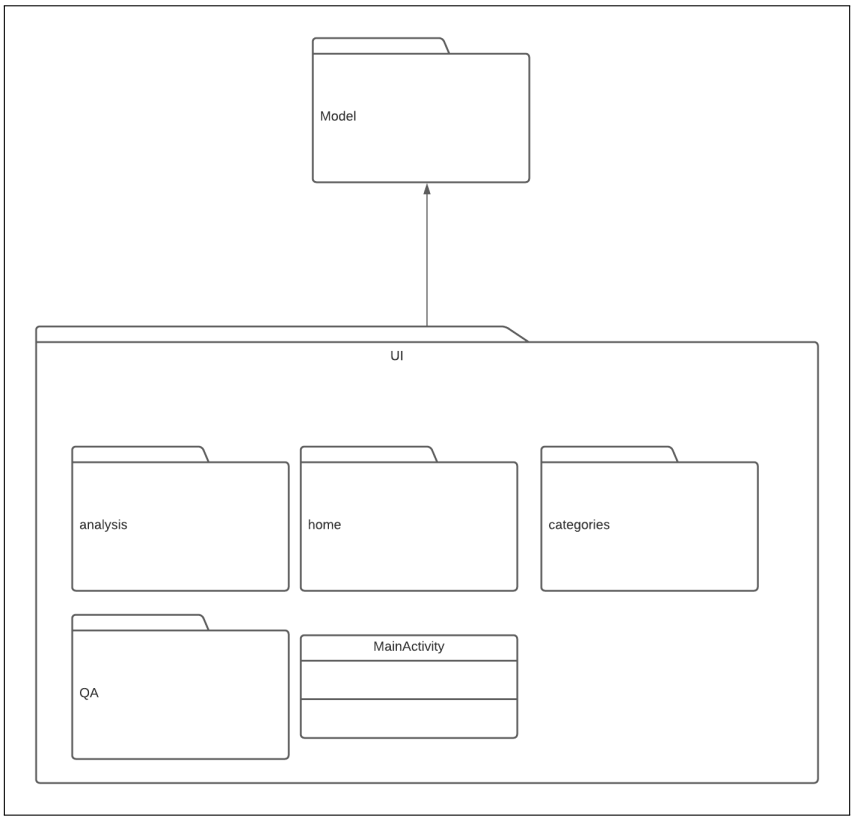

Figure 1: UML package diagram.

3.2 UML class diagram

3.2.1 model package class diagram

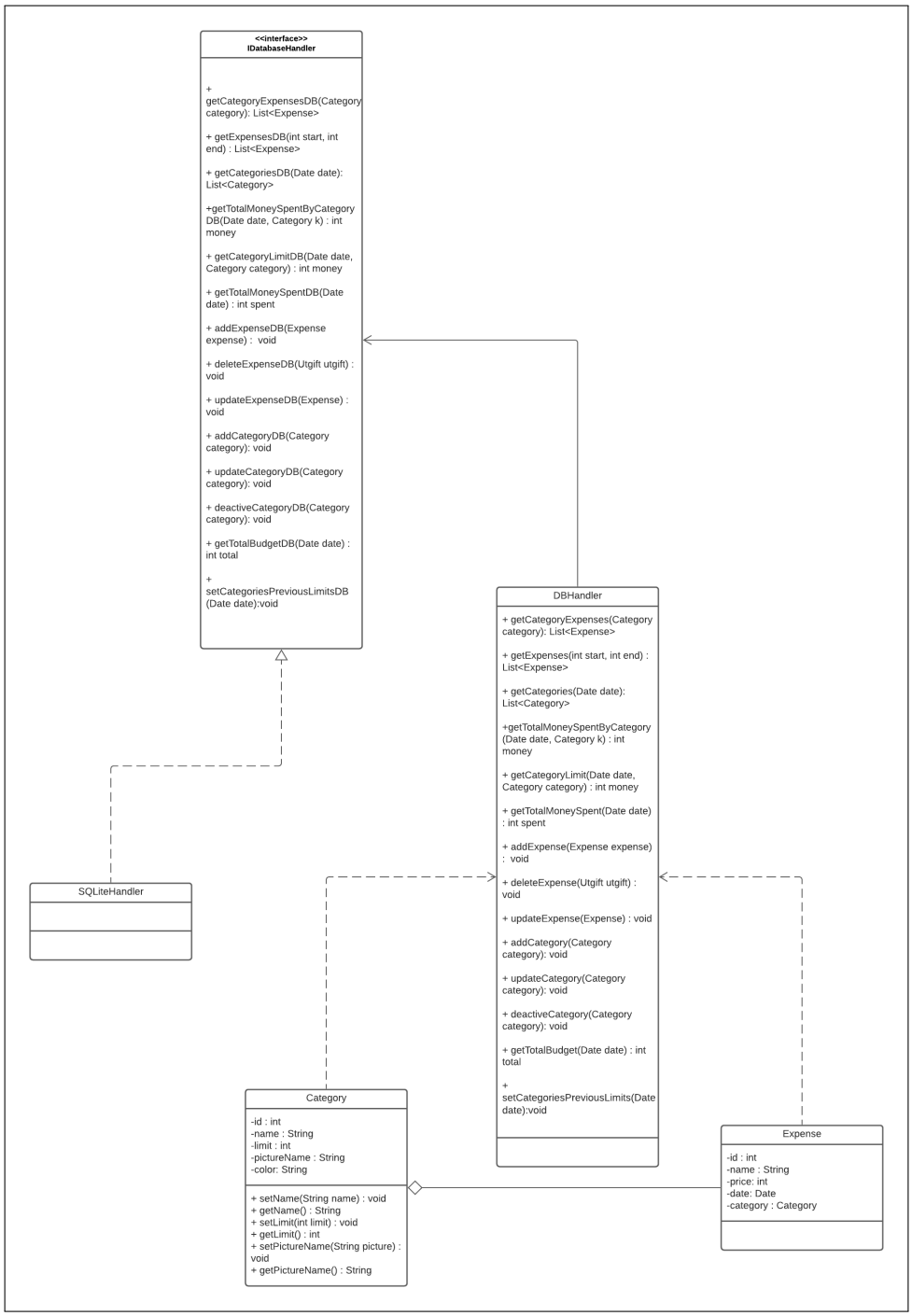

Figure 2: Model class diagram.

3.2.2 Home package class diagram

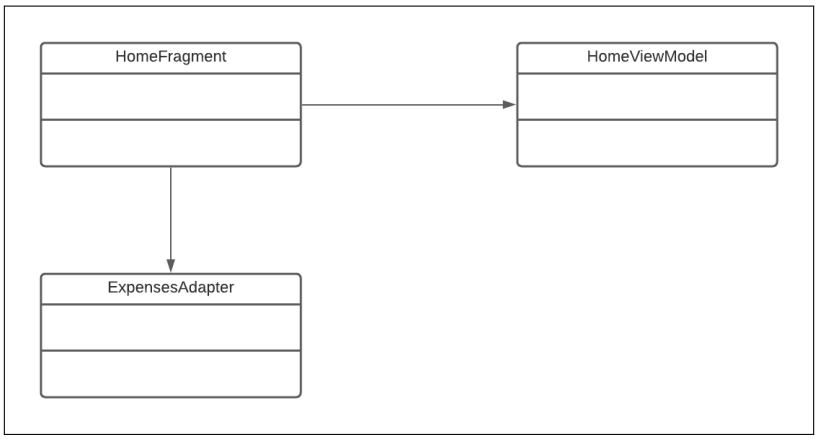

Figure 3: Home class diagram.

3.2.3 Analysis package class diagram

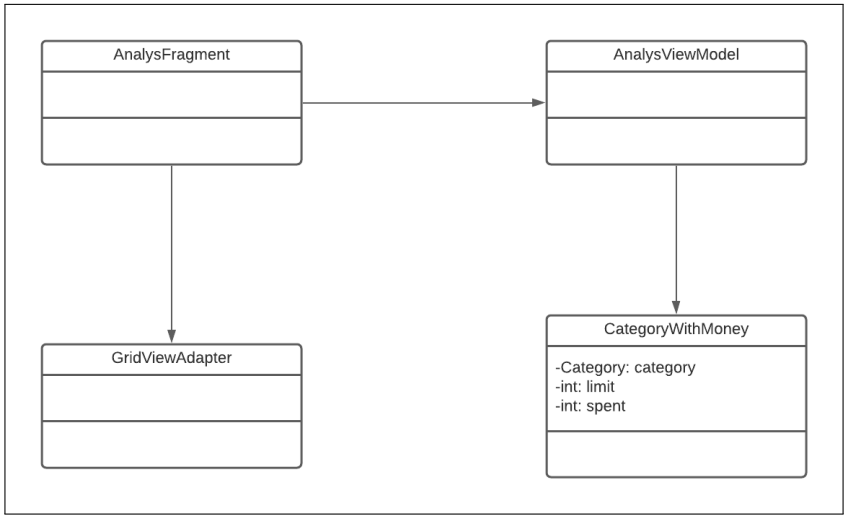

Figure 4: Analysis class diagram.

3.3 Sequence diagram

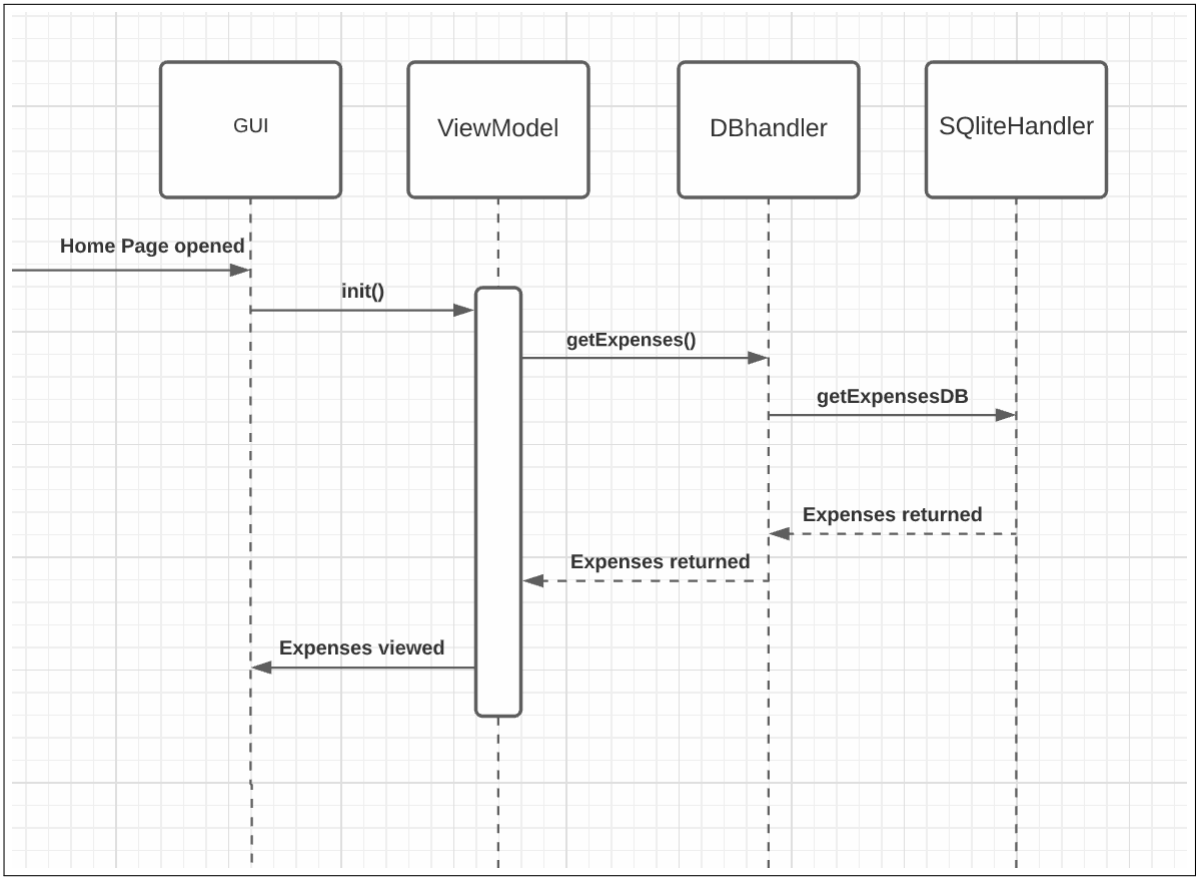

Figure 5: Home Sequence diagram.

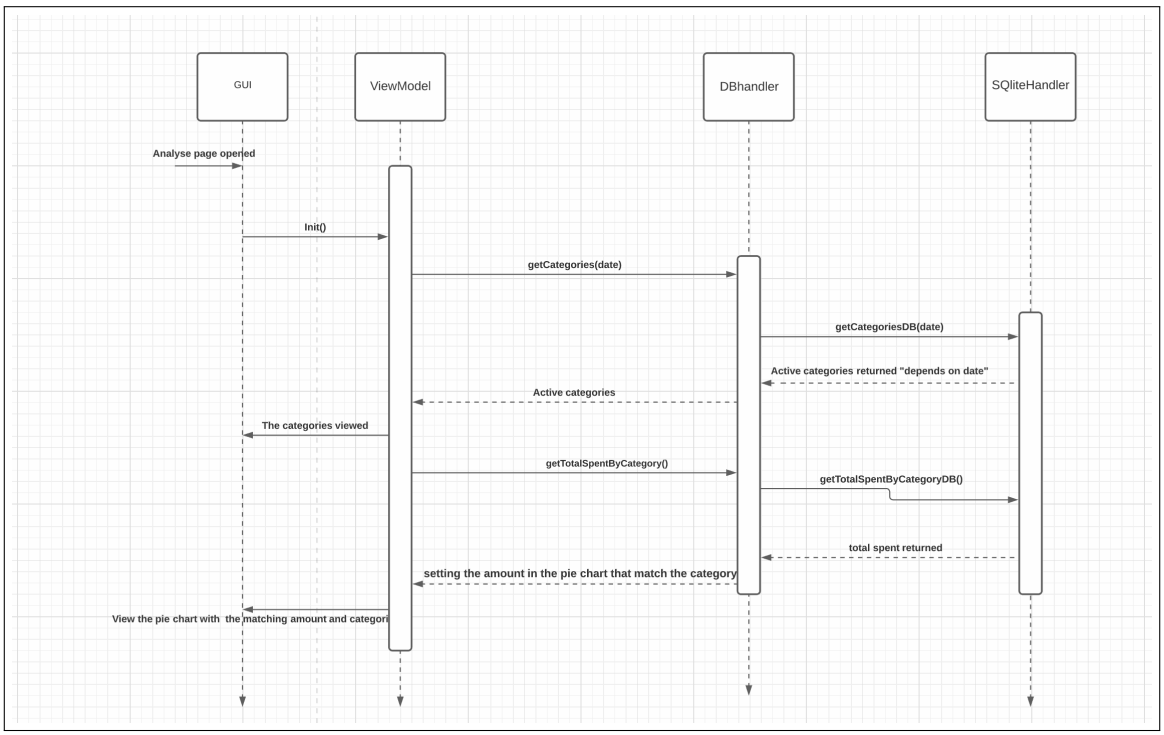

Figure 6: Analysis Sequence diagram.

3.4 Design patterns

Facade pattern: Facade pattern har använts för att ”dölja” services (SQLite) och minska klientens beroende av interna detaljer vilket leder till att kommunikationen blir mycket smidigare och minskar kopplingar mellan klienten och databasen. Detta har sket genom DBhandler klass som hämtar data från databasen och skickar den vidare till klient kod.
Model-View-ViewModel pattern: Applikationen följer MVVM patern. 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 och samtidigt lyssnar till ändringar som sker i UI för att uppdatera motsvarande data i databasen. Varje View klass kommer att ha sin egen ViewModel kass som har ovannämnda ansvar.
Bridge pattern: Bridge pattern använts för att förenkla bytet av databasen Vi har ett interface som har alla metoder som vår DBHandler behöver. SQLiteHandler klassen extend denna interfacet. DBhandler beror inte på SQLiteHandler(databasens klass) men har ett attribut av interfacet IDatabasHandler som pikar i sin tur på SQLite klassen. Detta ger oss möjlighet att byta databasen när som helst då det är bara att skapa en ny databasklass och byta referensen som attributen pikar på till den nya databasen.

4 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 kategroier 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.

5 Quality

Test till DBhandler är gjort men vi kan inte se coverage procent för någon anledning. Detta ska lösas i kommande iterationer. Testerna är gjorda med Junit och de finns i klassen ExampleInstumentedTest. PMD analys plug-in har använts för att undersöka om koden följer Java standard principer. Bild på resultat finns i referens, Figure 1.

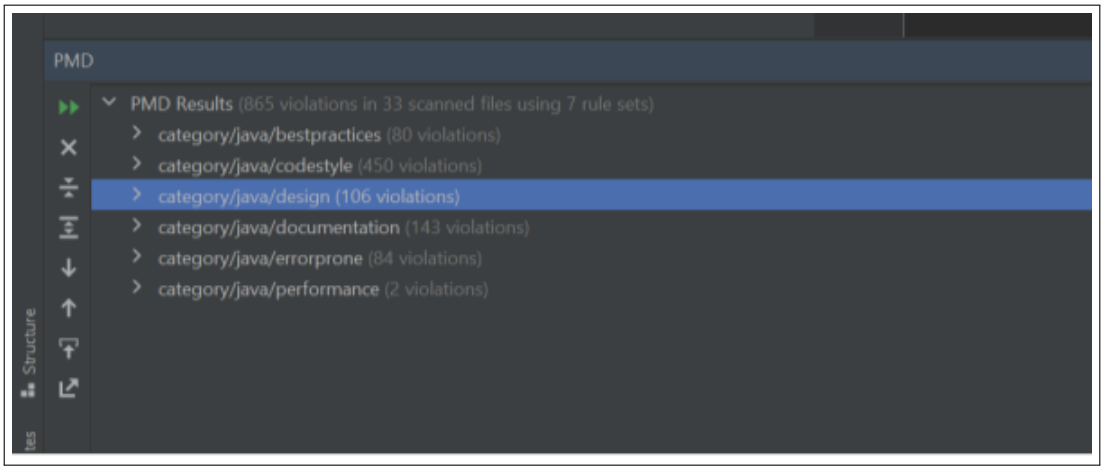

Figure 7: PMD.

6 References