2021

Spendio

MADE BY

HASAN KHALAF AND MATTIAS SPÅNGBERG

Mattias Spångberg: spma20pm@student.ju.se Hasan Khalaf: khha20qz@student.ju.se

Contents

Introduction 2

Architecture and planning 3

Architecture 3

Planning 3

Implementation 5

Graph 5

Hardware 5

Core Data 5

JSON parsing 6

Error handling 6

Tests 6

Learnings 7

Outlook 7

Spendio

Introduction

With every passing year more and more things require our focus financially. Maintaining your living expenses, hobbies, streaming services, presents, transportation and much more. This is just some of the things that needs funds. It's harder than ever to keep track of every expense. With Spendio on the other hand it's easy to keep track of all expenses. Add categories to easily manage your expenses and keep things tidy. Went abroad? You can add expenses under another currency and when you get home, view all expenses under your chosen currency so you really know how much you've spent. And if you don't want someone else prying into your spending's you can add authentication so that your information is safe.

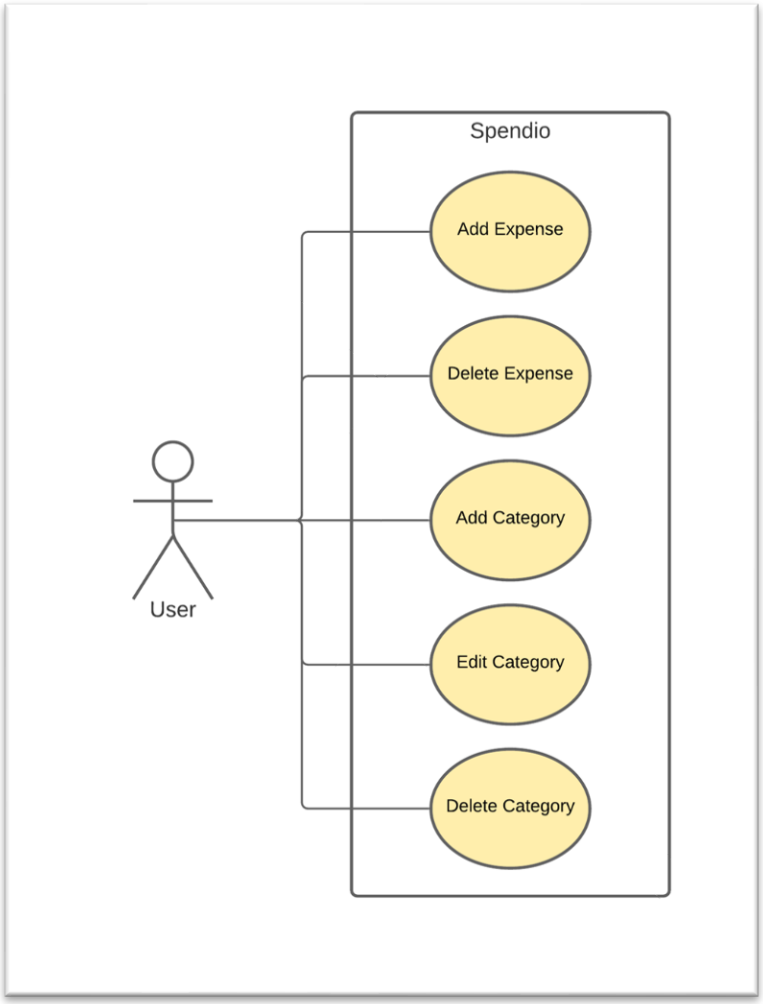

FIGURE 1: USE CASE DIAGRAM OF SPENDIO

Architecture

Before the implementation began, it was already decided to use Apples recommended pattern model-view-viewmodel (MVVM). This pattern is a client-side design that is effective especially on small-scaled applications. Figure 2 below shows the relationships between MVVM components.

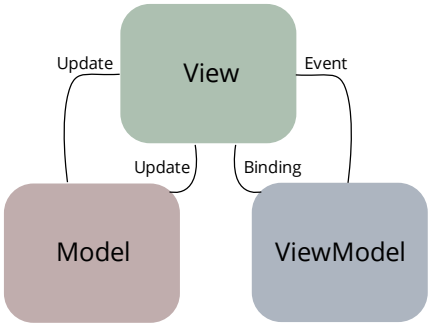

FIGURE 2: THE MODEL-VIEW-VIEWMODEL PATTERN

With this pattern it is easy to manage, troubleshoot and test the code, which is one of the reasons why it was chosen. Unit testing ensures that the application meets quality standards before it is deployed.

Planning

Spendio was planned with five different views. Every view had a specific purpose and all that could be displayed in multiple views was put in their own smaller screens so that they could reused in the five main views. An example of this is the expenseRowView that knows how the list of expenses should look like. Both Home View and History View use this smaller view to display expenses. The five major views can be seen in the mind map in figure 3.

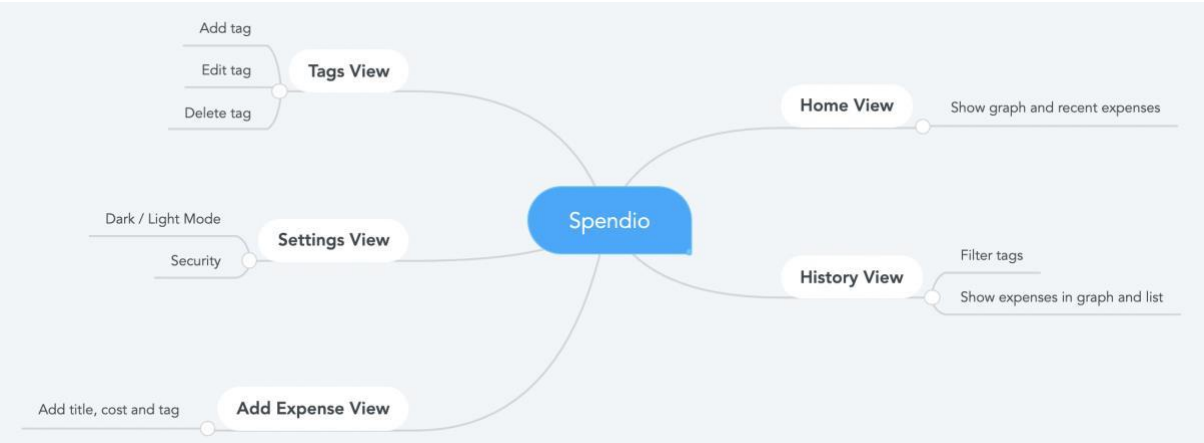

FIGURE 3: MIND MAP OF SPENDIO

As seen in figure 3, Spendio had plans early on to use Core Data as database. The plan was to have full CRUD (Create, read, update, delete) functionality so the user has full control over the elements in Spendio. Spendio was planned with a graph to show recent spending's and camera functionality so a user could add a picture to an expense. Both can be seen in figure 4.

FIGURE 4: PLANNED RECENT SPENDING'S PAGE FOR SPENDIO

Implementation

Implementing the different features for Spendio followed a direct course, some things had to be implemented before others. Categories came first as expenses needed it to function. A database was planned early on to store expenses in. Categories was therefore made with a database in mind. Same with expenses which was the next step. Having categories and expenses fixed and ready opened for a lot of branches in the workflow. The database could be implemented, error handling was ready to start and work on the graph begun.

Graph

The graph was an easy implementation. A package called “SwiftUiCharts” (version 1.5.7) was used to display graphs. This turned out to be a curse disguised as a blessing. The graphs where clean and easy to use. A function was made and took all expenses of a certain category and sent it to the graphView to render. At first, it worked fine but after some time a problem emerged. There was no way to update the graph view when a new value was added to the view. Therefore, after a lot of work, the graph was scrapped as it did not meet standards.

Hardware

As hardware, camera functionality was up for discussion. A user could add a picture to their expenses to further document what the expense was about. This was later scrapped as it would make the expenses hard to read and less user friendly. Adding color to a category was introduced instead so that users still could easily see what category an expense belonged to. Spendio got touch id as planned as hardware function. A user can turn on authentication in the settings menu to protect information from prying eyes.

Core Data

Spendio uses Core Data to organize data. Since the usage of Spendio is to be able to use it offline, Core Data without CloudKit seemed to be the right choice for this purpose. Figure 5 below shows a detailed overview of the Core Data tables.

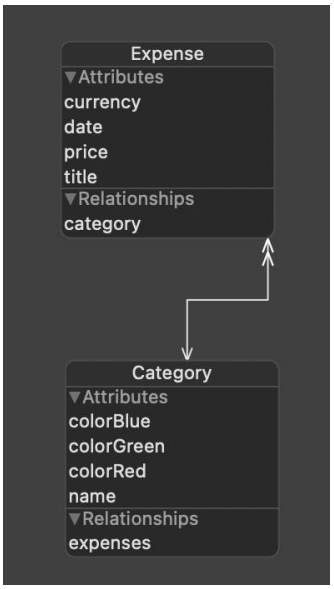

FIGURE 5: LOGICAL ENTITY RELATIONSHIP DIAGRAM OF THE CORE DATA TABLES

Resources that are stored in Core Data are expenses and categories. They are related to zero-to-many optional relationships, meaning that expense can be either uncategorized or belong to one category.

When it comes to implementing the Core Data, it was difficult to follow MVVM pattern. Instead, a controller was introduced to the architecture. Figure 6 below illustrates Core Data components.

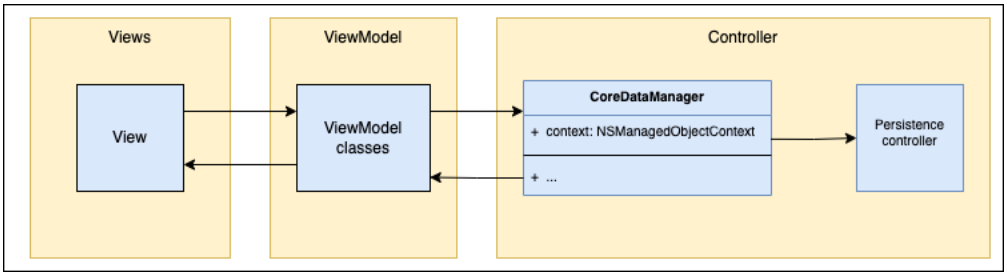

FIGURE 6: CORE DATA ARCHITECTURE OF SPENDIO

JSON parsing

As JSON parsing and networking Spendio fetches data from an application programming interface (API) on the web called “freecurrencyapi”. Spendio uses this API to fetch exchange rates for a user’s selected base currency. A function takes all expenses and converts them to the selected currency. If a user buys things on vacation and later wants to see how much they spent in their native currency, they can select it as their base currency. All expenses now show in that currency.

Error handling

To handle errors in a smooth way it was necessary to think of a smart solution. Therefore, the solution was to create an error class that follows singleton pattern which can be accessed anywhere through the application. If any error occurs, for example while fetching data from API or Core Data, a popup screen is displayed to inform the user of the problem.

The popup screen itself is a view modifier that activates when an error occurs, in other words It observes the error class.

Tests

When it comes to tests, Spendio has four-unit tests. Two for form validation on categories and expenses, and two for Core Data for create and delete category. There was important that the database was tested as it was a big part of Spendio. If the database did not function properly, uses would not see their expenses and basic functionality break.

Learnings

Using a declarative language after a year of imperative programming was a challenge. Thinking of different states and how to change between them depending on the situation was new but after working on it for a while the lesson stuck.

There were a lot of learnings when it comes to design patterns. Following Apples recommended MVVM design pattern, Spendio is clean and easy to understand. Spendio has a lot of models and view models that all serve a single purpose to keep high cohesion in the project. The focus of Spendio was a code base that stuck to MVVM, was clear and easy to navigate. The design pattern helped a lot with that.

Outlook

Spendio is built for expense management and does so well. But some functions that would add to the user experience is a graph and more robust filter functions. If Spendio was to be developed more there would be built in graph functions that could display different graphs depending on the current filter. Currently Spendio supports four different currencies, but more could easily be added and would give more reason for users to use the app. Spendio has a robust database that keeps track of all that is to know about an expense. It is already prepared for filter functions that involve dates, prices and more. Spendio is built with the future in mind and is therefore just a small step from further implementations.

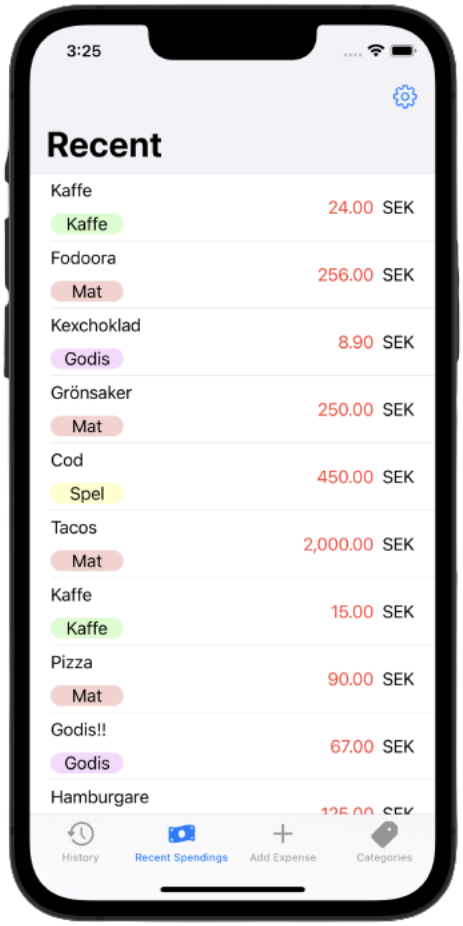

FIGURE 7: FINAL RECENT PAGE FOR SPENDIO