Overview

The main aim of this coding exercise is for us to understand the way you approach a task. How you break down the problems and how you apply standard design patterns within your code will be examined. The better you document your design and implementation decisions along the way, the better we can achieve the goal of this exercise.

Much of the work within Moneytree requires making use of a network resource and processing the information provided. This challenge will involve the same ideas that you will be working with in Moneytree.

The data story is mandatory, and of the remaining user stories, please pick two to do.

We expect you to spend 3 to 6 hours on this exercise. It doesn’t have to be done all at once, and you’re not expected to finish everything.

Introducing the Moneytree Light app

The Moneytree Light app displays accounts and their transactions for Japanese users. All of the JSON files are currently included in the bundle for prototyping, but in the imaginary future development roadmap, we are planning to fetch the JSON from a server.

There are two screens. The initial view displays a list of accounts and its total balance of those accounts. The top 1/5 of the screen shows the total amount, while the rest of the screen displays the list of accounts. The list of accounts is grouped by its institution name. Each item in the list shows the nickname of the account and the balance in the account's original currency, and should be ordered based on the account's nickname.

Tapping an account loads a second view where the list of transactions belonging to that account are displayed. The top 1/5 of the screen displays information about the account, including the balance in the user's local currency (JPY). The transactions list is grouped by the month, and each row displays the date, description and amount in its account's currency.

Data Story

3 JSON payloads are included in the bundle. Accounts are in accounts.json, and for each account, there is a corresponding transaction_#<account_id>.json file that contains transactions. The file size of this JSON can be quite large, so it might take some time to load and parse. The user interface should not be blocked during this process.

You can choose to keep the returned information in memory or store the data. If you store the data then the app should load the data from storage when launched. The type and manner of storage is left to your discretion.

User Stories

Assumption: The user is Japanese, and their primary currency is the yen.

1. User should be able to see a list of accounts and the total balance of those accounts in the user's currency. The list should be ordered by the account's nickname and grouped by its institution. Each row should display the nickname and the balance in that account's currency.

2. User should be able to tap on the account to see the list of transactions on that account. The transactions should be ordered by month, from the newest one to the oldest. Each row on the transactions list should display the transaction date, description and its amount in the account's currency.

3. In the transaction list view, a user should see the transactions grouped by month, with each of the header display the month and the year including its flow (money in and money out) during that month.

4. In the transaction list view, a user should be able to see the account details, like its nickname, balance, total transactions in and out in a current month, including its estimate balance in the local currency.

5. As a user, I should be able to search for transaction in the list based on the description or the amount. When searching, I should be able to tap anywhere to cancel the search

6. As a user, I should be able to delete a transaction from the transactions list after confirming that I really want to delete the transaction.

Instructions

Complete the data story and at least 2 user stories.

This project should be written in either modern Objective-C or Swift (or both). Modern Objective-C means methods are annotated with nullability keywords, classes have a designated initialiser, and lightweight generics are used whenever appropriate. Almost half of the code bases in use at Moneytree are currently Objective-C, while the newer ones have been written in Swift. So you will need knowledge of both languages at Moneytree.

Please track your work in git.

There are screenshots placed in the "ideas" folder. Don't worry about the tab and the user interface. The UI should be built with standard layouts and controls. At Moneytree, we prefer to build UI in code, rather than using Interface Builder. However, feel free to use whichever you are most comfortable with for this exercise.

Some things we're looking for

- **Understanding how you work**
  Communicate your decisions to us, through well-designed, simple code and a README file, documenting your overall design, as well as any decisions you made through the implementation process. Additionally, since this exercise is time-constrained, feel free to share the next steps you would take if you had more time.

- **No warnings**
  We love code that compiles without any warnings. But we understand that's not always possible. If you can't resolve a particular warning, explain why, and how you attempted to fix it.

- **Testing**
  Unit or Integration tests are required and we also love good test coverage!

- **Accessibility**
  Making an accessible app is often overlooked. We will take special note of any efforts to support accessibility features.

Impress us with your understanding of the challenge and show us your coding personality!