Outline of Technical Specifications for Trackify (Update)

Sadiq Al-Humood, Joseph Lee
December 5, 2024

Outline of the Tech Stack:

- Platform: Android, Jetpack Compose for flexible user interface integration and for composing our user interface design and layout
- We will be using Kotlin to curate our expense tracking application
- Incorporate Room database for local data storage; useful for storing information regarding the user's bank account details, points granted, photos of receipts, transactions, etc…
- Utilize Retrofit with Coroutine support to facilitate API requests
- Firebase Cloud Messaging to send push notifications to the user regarding progress on the monthly budgetary goal
- Firebase Firestore: cloud-based database that will be employed to handle real-time data syncing for the application's leaderboard game system. It will store leaderboard data, the user's points, and update rankings in real-time, although this process requires internet connection. Synchronization across all devices and all users will be possible through this database.
-

Required APIs:

- Firebase Authentication - for secure user authentication.
- Teller API - Securely connects to user bank accounts and fetches real-time account and transaction data; integrated via **Firebase Functions** to handle mTLS and HTTP Basic authentication, ensuring secure communication with the Teller API. Accessed to display transaction data, analyze savings, and track user expenses.
- Google Cloud Vision API - to process images of receipts, extracting text data to simplify expense tracking; data from photos will be stored in a Room database local to the user

External Libraries:

- Compose Charts: For building dynamic graphs and visualizations, such as savings trends from the previous month. Data for charts will be stored locally in the Room database.

- Retrofit (Firebase Functions) - Makes HTTP requests to Firebase Functions endpoints which fetches transaction and account data through Teller API integration via Firebase Functions.

Device Sensors:

- Touch Screen: for essential interactions like tapping, swiping, and scrolling through the app.
- Camera:To capture receipt images, enabling the app to extract transaction data and store it in the Room database.

Database Schema:

- Firebase Firestore: cloud-based database that will be employed to handle real-time data syncing for the application's leaderboard game system. It will store leaderboard data, the user's points, and update rankings in real-time, although this process requires internet connection. Synchronization across all devices and all users will be possible through this database.
- Room Database for local storage for the user (expenses, receipts, budget configurations, savings); ensures offline functionality
- Firebase Cloud Messaging: enables push notifications to remind users of monthly budgetary goals and their savings progress from the previous month.

Database Schema for Room:

transactions_table
  ├── id: Int (Primary Key)
  ├── name: String
  ├── date: Date
  ├── amount: Double
  ├── category: String
receipts_table
  ├── id: Int (Primary Key)
  ├── imageUri: String
  ├── associatedTransactionId: Int (Foreign Key)
savings_table
  ├── month: String
  ├── totalSavings: Double
  ├── goal: Double

Firestore Database Schema:

```
/users
  └── {userId}
      ├── accountId: String
      ├── accessToken: String
      ├── savingsGoal: Double
      ├── rewards: List<Object>
      └── transactions: Collection
          ├── {transactionId}
              ├── name: String
              ├── amount: Double
              ├── date: Timestamp
              └── category: String
/leaderboard
  ├── {userId}
      ├── totalSaved: Double
      └── rank: Int
```

User Journey:

- Users log in using Firebase Authentication
- Bank accounts are linked via Teller Connect, with Firebase Functions fetching transaction data securely.
- Users set monthly savings goals, and transactions are categorized for tracking.
- Real-time leaderboard rankings and rewards incentivize savings.
- Formula for Point - System: Points = SavingsPercentage * Multiplier
    - SavingsPercentage: The percentage of income or spending that the user saved during the 30-day period.
    - Multiplier: A scaling factor to make the point system proportional to the savings percentage. We will make this 10 for every user.
    - No loss for failing to achieve the user's budgetary goal, but no gain either. Net zero gain.
    - Example) The user saved 20% of their income in a 30-day period. He/she will receive 20x10 = 200 points