**Douglas College**

**CSIS 3175-003**

**Introduction to Mobile App Dev**

**Personal Finance Application – TRExpense**

**Project Report**

Submitted by:

**Aakash - 300374586**

**Janvi Arora - 300383801**

**Vu, Tung - 300386998**

Submitted to: **Xing Liu**

GitHub link: https://github.com/aakash2019/TrExpense

December 1st , 2024

# Introduction:

The Personal Finance Manager application which is designed to help users efficiently track

their income, expenses, and savings, providing a comprehensive view of their financial

health.

User-friendly interface, users can easily input their spending, categorize expenses, and set

monthly budget goals. Additional features include budget planning aims to empower users

to make informed financial decisions and better manage their finances.

## Why do we need a personal finance tracking application?

Managing personal finances is a crucial aspect of modern life. A personal finance tracking application provides tools to address several challenges faced by individuals in organizing their financial lives. Below are the key reasons why such applications are necessary:

* Understanding Financial Habits
* Budgeting Made Simple
* Encouraging Savings
* Avoiding Debt
* Time Efficiency
* Goal Tracking
* Currency Management
* Data-Driven Decision Making
* Accessibility and Convenience
* Environmental Awareness

# Project Timeline and Planning

A blue and white graph

Description automatically generated

## Week 1: Project Idea Proposals

* To brainstorm and finalize a feasible, impactful, and practical project idea.
* Ensures alignment with objectives, available resources, and the timeline.
* Provides clarity on the project's scope and deliverables.

## Week 2: Welcome Screen and Basic Wallet Functions

* **Welcome Screen (Create Wallet, Show Wallet):**
  + Introduces users to the application and ensures a smooth onboarding process.
  + Allows users to set up their wallets, a foundational step for the app's functionality.
  + First impression matters; this step ensures the app is user-friendly from the start.
* **Basic Functions (Add/Remove Expenses, Categorize Income/Expenses):**
  + Core functionality of the app; allows users to track their expenses and income.
  + Categorization helps in generating meaningful reports and insights.
  + Provides immediate value to users by addressing their primary need.

## Week 3: Progress Presentation

* Opportunity to gather feedback on the progress made so far.
  + Ensures alignment with project goals and addresses potential gaps early.
  + Helps validate the app’s functionality, user experience, and design with other team members.

## Week 4: Additional Features

* **Budget Creation:**
  + Enhances the app's value by enabling users to plan and control their spending.
  + Helps users set financial goals, fostering long-term engagement.
* **Monthly Overview:**
  + Provides users with summarized insights about their financial activities.
  + Encourages better financial management by showing trends and areas for improvement.

# Possible deliverables:

1. Function to create a wallet in different currency.

2. Presentation of wallet with relevance views.

3. Add an expense/income.

4. Spending Alerts.

5. Create a saving/budget.

6. Show graphical expense/income based on weeks/months.

7. Testing cases and result

8. User documentation

# Features and Functions

## Firebase Authentication:

### Sign in:

A screen shot of a phone

Description automatically generated

**1. Sign-In Functionality (LoginActivity.java)**

The **sign-in process** is implemented in the LoginActivity class, which authenticates users using Firebase Authentication. The class has the following core components:

**1.1 UI Components**

* **Email and Password Inputs**: Users input their credentials (emailInput and passwordInput).
* **Login Button**: Initiates the login process.
* **Sign-Up Text**: Redirects users to the sign-up page.
* **Forgot Password Text**: Placeholder for password recovery.

**1.2 Firebase Integration**

* Firebase Authentication (firebaseAuth) and Firestore (db) are initialized to manage user accounts and associated data.

**1.3 Functionality**

* **Automatic Login Check**: If a user is already signed in (currentUser is not null), the app bypasses the login screen and checks the user's wallet using checkWallet(FirebaseUser user).
* **Login Button**: Calls the loginUser() method to authenticate credentials.

**1.4 The loginUser() Method**

* Retrieves user input for email and password.
* Validates the inputs, ensuring neither is empty.
* Authenticates credentials using Firebase’s signInWithEmailAndPassword().
* On success, it fetches the current user and calls checkWallet() to verify if the user has an existing wallet.

**1.5 Wallet Verification (checkWallet())**

* Queries Firestore to check if the user has wallet documents in their collection.
* If a wallet exists, the user is redirected to TabsActivity. Otherwise, they are directed to CreateWalletActivity.

**1.6 Error Handling**

* Displays messages using Toast to inform the user about login failures or issues retrieving wallet data.

### Sign Up:

A screen shot of a phone

Description automatically generated

**2. Sign-Up Functionality (SignUpActivity.java)**

The **sign-up process** is handled in the SignUpActivity class, enabling users to create accounts and initialize their profiles in Firebase.

**2.1 UI Components**

* **Input Fields**: For first name, last name, email, and password.
* **Currency Spinner**: Allows users to select their preferred currency.
* **Sign-Up Button**: Triggers the account creation process.

**2.2 Firebase Integration**

* Firebase Authentication (mAuth) handles user creation.
* Firestore (db) stores user profiles.

**2.3 Functionality**

* **Currency Selection**: The setupCurrencySpinner() method initializes the spinner with a list of currencies. A default selection (Canadian Dollar) is provided, and the selected value is stored in selectedCurrency.

**2.4 The signUp() Method**

* Validates user input to ensure required fields are filled.
* Creates a new user using Firebase’s createUserWithEmailAndPassword().
* On success:
  + Retrieves the user's unique ID (userId).
  + Constructs a UserProfile object with the user's details.
  + Saves the profile to Firestore under the users collection.
  + Redirects the user to CreateWalletActivity to set up their wallet.
* On failure, it displays an error message.

**2.5 UserProfile Class**

* A helper class representing user data (first name, last name, email, currency, and user ID) for Firestore.

**2.6 Error Handling**

* Ensures all required fields are filled before proceeding.
* Displays Toast messages for sign-up failures or Firestore write issues.

### Summary of Sign-In and Sign-Up

| **Aspect** | **Sign-In** | **Sign-Up** |
| --- | --- | --- |
| **Purpose** | Authenticate existing users. | Register new users and create profiles. |
| **Primary Method** | loginUser() | signUp() |
| **Firebase Usage** | Verifies user credentials. | Creates new user credentials and profile. |
| **UI Flow** | Checks wallet and redirects accordingly. | Initializes profile and prompts wallet setup. |

## Main Page:

### Home Fragment:

A screen shot of a cell phone

Description automatically generated

**1. HomeFragment**

The HomeFragment displays an overview of the user's wallets and their balances, integrating Firebase Firestore for data management.

**1.1 Features**

* **Total Balance Display**: Calculates and shows the user's total wallet balance in Canadian Dollars (CAD).
* **Wallet List**: Dynamically lists wallets with their respective names, balances, and currencies.

**1.2 Firebase Integration**

* **Fetch Wallet Data**: Uses Firestore to query the user's wallet collection and retrieve wallet details (name, balance, currency).
* **Real-time Updates**: Fetches data upon fragment loading.

**1.3 Key Methods**

* fetchWalletData(): Queries Firestore for wallets and accumulates the total balance.
* addWalletToLayout(): Dynamically creates UI components for each wallet using a LinearLayout and updates the walletListLayout.

**1.4 Error Handling**

* Logs errors during data retrieval and ensures the UI remains responsive even if Firestore queries fail.

### Transaction Fragment:

A screenshot of a cell phone

Description automatically generated

**2. TransactionsFragment**

The TransactionsFragment allows users to view and manage transactions within a selected wallet.

**2.1 Features**

* **Wallet Selection**: Users can select a wallet from a dialog to view its transactions.
* **Transaction Display**: Lists all transactions associated with the selected wallet.
* **Add Transaction**: Redirects users to a transaction creation screen (AddFragment).

**2.2 Firebase Integration**

* **Fetch Transactions**: Queries Firestore for transactions within a specific wallet.
* **Wallet Management**: Updates the currentWallet field in Firestore based on user selection.

**2.3 Key Methods**

* loadCurrentWallet(): Retrieves the user's current wallet and fetches its transactions.
* fetchTransactionsForWallet(): Queries transactions for the selected wallet and populates the UI.
* showWalletSelectionDialog(): Displays a dialog for selecting a wallet.
* navigateToAddTransactionFragment(): Navigates to the AddFragment.

**2.4 Error Handling**

* Displays Toast messages for missing wallets or transaction-related issues.

### Add Transactions Fragment:

A screenshot of a cell phone

Description automatically generated

**3. AddFragment**

The AddFragment facilitates the creation of new transactions for a selected wallet.

**3.1 Features**

* **Transaction Details**: Users can input transaction amount, category, date, and an optional note.
* **Wallet Selection**: Allows users to select a wallet for the transaction.
* **Validation**: Ensures all fields are properly filled before submission.

**3.2 Firebase Integration**

* **Add Transaction**: Creates a new transaction document in Firestore under the selected wallet's collection.
* **Update Wallet Balance**: Deducts the transaction amount from the wallet's balance.

**3.3 Key Methods**

* fetchCurrentWallet(): Retrieves the current wallet to initialize the transaction form.
* uploadTransaction(): Validates inputs, creates a transaction in Firestore, and updates the wallet's balance.
* openCategoryDialog(), openNoteDialog(), openDatePickerDialog(): Dialogs for selecting or entering transaction details.

**3.4 Error Handling**

* Checks for sufficient wallet balance and validates all required inputs before submission.

### Profile Fragment:

A screen shot of a cell phone

Description automatically generated

**4. ProfileFragment**

The ProfileFragment displays user information and provides options for wallet and account management.

**4.1 Features**

* **User Profile**: Fetches and displays the user's name and total wallet balance.
* **Wallet Management**: Allows users to create or edit wallets.
* **Logout**: Provides functionality to sign out of the account.

**4.2 Firebase Integration**

* **Fetch User Data**: Queries Firestore for the user's profile and wallets.
* **Wallet Operations**: Redirects to activities for creating or editing wallets.

**4.3 Key Methods**

* fetchUserProfile(): Retrieves user details and wallet balance.
* fetchWalletData(): Aggregates wallet balances to display the user's total balance.
* createNewWallet(), editWalletButton(): Handles wallet creation and editing.
* logoutUser(): Signs out the user and redirects to the login screen.

**4.4 Error Handling**

* Ensures robust feedback through Toast messages in case of Firestore errors or missing data.

### Summary of each Fragment:

**Summary Table of Fragment Responsibilities**

| **Fragment** | **Responsibilities** | **Firebase Usage** |
| --- | --- | --- |
| **HomeFragment** | Displays wallet balances and total balance. | Queries wallet collection. |
| **TransactionsFragment** | Displays and manages transactions for the selected wallet. | Queries wallet and transaction collections. |
| **AddFragment** | Creates new transactions and updates wallet balances. | Adds to transaction collection, updates wallet balance. |
| **ProfileFragment** | Manages user profile and wallet operations. | Fetches user data, wallet information, and supports wallet editing. |

## Wallet Activities:

### Create Wallet Activity:

A screen shot of a phone

Description automatically generated

**1. CreateWalletActivity**

The CreateWalletActivity allows users to create a new wallet and save its information in Firebase Firestore.

**1.1 Features**

* **Wallet Creation**: Users can input wallet details, such as name, initial balance, and currency.
* **Set as Current Wallet**: The newly created wallet is automatically set as the user's current wallet.

**1.2 Firebase Integration**

* **Firestore Collection**: The wallet is stored under the user's wallets collection in Firestore.
* **Current Wallet**: Updates the currentWallet field in the user's Firestore document.

**1.3 Key Methods**

* setupCurrencySpinner(): Initializes a dropdown menu for selecting the wallet's currency, defaulting to CAD.
* createWallet():
  + Validates the wallet name.
  + Generates a unique wallet ID using UUID.
  + Creates a wallet document in Firestore and sets it as the user's current wallet.

**1.4 Error Handling**

* Validates wallet name input and provides error messages using Toast for empty fields or Firebase errors.

**1.5 Workflow**

1. User enters wallet name and balance, selects a currency, and clicks the "Create Wallet" button.
2. Wallet data is saved to Firestore, and the wallet is set as the current wallet.
3. User is redirected to the TabsActivity.

### Edit Wallet Activity:

A screen shot of a cell phone

Description automatically generated

**2. EditWallet**

The EditWallet activity enables users to update wallet details or create a new wallet if none exists.

**2.1 Features**

* **Edit Existing Wallet**: Users can update the wallet name, balance, and currency.
* **New Wallet Creation**: If no wallet ID is provided, a new wallet is created.

**2.2 Firebase Integration**

* **Update Wallet**: Updates an existing wallet document in Firestore.
* **New Wallet**: Creates a new wallet document if no wallet ID is passed.

**2.3 Key Methods**

* setupCurrencySpinner(): Similar to CreateWalletActivity, initializes a dropdown for currency selection.
* loadWalletData(String walletId):
  + Fetches wallet details from Firestore using the wallet ID.
  + Pre-fills the input fields with retrieved data.
* saveOrUpdateWallet(String walletId):
  + Validates wallet name input.
  + Updates an existing wallet or creates a new wallet in Firestore based on the wallet ID.

**2.4 Workflow**

1. If a wallet ID is provided:
   * Wallet data is fetched and pre-filled in the input fields.
   * User makes edits and saves changes.
2. If no wallet ID is provided:
   * A new wallet is created with the entered details.
3. Upon success, the user is redirected to TabsActivity.

**2.5 Error Handling**

* Displays Toast messages for:
  + Missing wallet name input.
  + Firebase errors during wallet creation or update.
* Ensures authenticated users before performing Firebase operations.

### Summary of Wallet Activities:

| **Activity** | **Purpose** | **Firebase Usage** |
| --- | --- | --- |
| **CreateWalletActivity** | Creates a new wallet and sets it as the current wallet. | Adds wallet to wallets collection, updates currentWallet field. |
| **EditWallet** | Updates existing wallet details or creates a new wallet. | Fetches, updates, or creates wallet documents in Firestore. |