

[Description](#)

[Intended User](#)

[Features](#)

[User Interface Mocks](#)

[Login Screen](#)

[Signup Screen](#)

[Main Screen](#)

[Add Expense Screen](#)

[Widget](#)

[Key Considerations](#)

[Android Studio](#)

[Programming Language](#)

[How will your app handle data persistence?](#)

[Describe any corner cases in the UX.](#)

[Accessibility](#)

[Resources](#)

[WorkManager API](#)

[Describe any libraries you'll be using and share your reasoning for including them.](#)

[Describe how you will implement Google Play Services.](#)

[Next Steps: Required Tasks](#)

[Task 1: Project Setup](#)

[Task 2: Configure Google sign-in](#)

[Task 3: Implements UI](#)

[Task 4: Setup Data layer](#)

[Task 5: Setup UI layer and uses cases](#)

[Task 6: Setup dependency injection](#)

[Task 7: Polish UI](#)

**GitHub Username:** [karan](#)

# Expense Tracker

## Description

Keep track of your everyday expenses all at one place. This app will definitely help you to find your extra expenditure that you can cut off and use your money efficiently. The user can add data manually and get to see the day's expense with a total at one place.

## Intended User

This application can be handy to all those who find it difficult to track their expenses using traditional methods. It can be useful to a student or an adult. Easy to use, simple and friendly user interface that can be used by anyone.

## Features

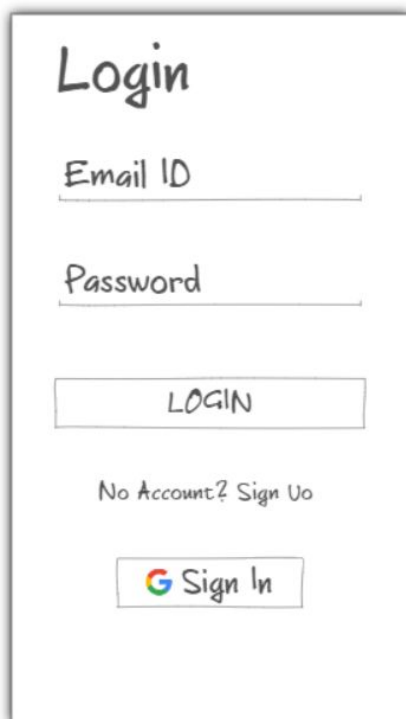
List of main features of the app:

- Firebase User Authentication.
- Add your expense after every payment or expenditure.
- Saves the data .
- Shows daily expense.
- Home Screen widget to add expenses on the go.

## User Interface Mocks

### Login Screen

Login screen for the user to login with email and password or can opt to sign in with google.



A hand-drawn mockup of a login screen. It features a title 'Login' at the top. Below the title are two input fields: 'Email ID' and 'Password'. Under the 'Password' field is a 'LOGIN' button. Below the button is a link that says 'No Account? Sign Up'. At the bottom is a 'Sign In' button with the Google logo icon.

## Signup Screen

A sign in screen to enter username and password and register the user using firebase authentication. Users can navigate to this screen from the login screen and also back to the login screen.

← Register

Email ID

Password

SIGN UP

## Main Screen

Main screen is where the user will be sent after a successful login. Users can add their expenses and also keep its track here. It shows a total of the expenses made by the user. New expenses can be added using the Floating Action Button at the bottom right corner.



## Add Expense Screen

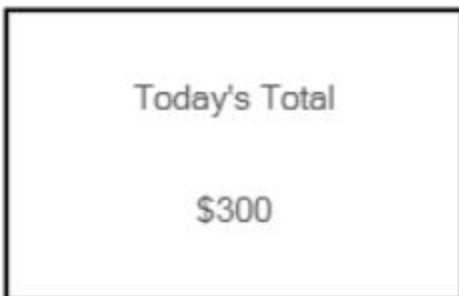
A screen to add the description and value of the expenditure. A toast will be shown after the successful addition of the expense.



A mobile app screen titled "Add Expense". At the top left is a back arrow icon. Below the title are two text input fields: the first is labeled "Description" and the second is labeled "Value". At the bottom center is a rectangular button labeled "ADD".

## Widget

A widget that shows the user's total expense of the day. Clicking on it will open the app's main screen.



A rectangular widget with a black border. It contains the text "Today's Total" in a medium font size, and below it, the text "\$300" in a larger font size.

## Key Considerations

### Android Studio

The project will be completed using Android Studio (version 3.5.1).

### Programming Language

The app will be written solely in Java programming language.

### How will your app handle data persistence?

Expense data will be stored locally using Room from Android Architecture components. Shared preferences will be used to store other necessary information.

### Describe any edge or corner cases in the UX.

App will show a no expense view until the user enters some data to notify that there is no data added.

### Accessibility

The app will support Android's accessibility features, through content descriptions.

### Resources

Values will be stored in a strings.xml file in res/values for Strings values along with RTL layout switching on all layouts. This will hold true for dimensional values, integers and booleans, all in their respective folders.

### WorkManager API

The app will use WorkManager API to allow users to schedule some tasks such as saving app data to cloud (Google Drive).

Describe any libraries you'll be using and share your reasoning for including them.

Libraries (with stable versions)	Descriptions
<b>Navigation (version: 2.2.2)</b>	to handle in-app navigation with ease. No need to do fragments transactions.
<b>LiveData (version: 2.0.1)</b>	to build data objects that notify views when the underlying database changes.
<b>Databinding (version:3.5.0)</b>	to declaratively bind observable data to UI elements.
<b>LifeCycle (version: 2.2.0)</b>	to make UI automatically respond to lifecycle events.
<b>Room Database (version: 2.2.5)</b>	to store our expenses data without SQLite boilerplate code.
<b>Dagger 2 (version: 2.22.1)</b>	for dependency injection
<b>Google open-sources-licences (version: 16.0.2)</b>	to display open-source licence notices in the app.
<b>Firestore Auth (version: 16.2.1)</b>	to authenticate users using Firebase.
<b>Google adMobs services (version: 17.2.0)</b>	to show ads in the app.
<b>Material components (version: 1.1.0)</b>	to make look and feel of the app different.

- **Gradle (version: 3.4.1)**
- **Support library: AndroidX**

Describe how you will implement Google Play Services or other external services.

- **Firestore Auth** will be used to let users sign in to our app using their Google account.
- **Google adMobs services** to display banner ads within the app.

## Next Steps: Required Tasks

### Task 1: Project Setup

- Create the project on Android Studio
- Add gradle dependencies for third-party libraries
- Create the app structure by adding packages for app components.

### Task 2: Configure Google Sign-in

- Create the project in Firebase console
- Configure the authentication flow.

### Task 3: Implement UI for Each Activity and Fragment

- Build UI for MainActivity to display current state of the users account: total spending of the day and a list of the most recent expenses.
- Build UI for the add expense fragment.
- Create a widget for the home screen.

### Task 4: Setup Data layer

- Create data classes.
- Create Room database components: DAO, database.
- Create an expense data repository to expose data to our viewModel.

### Task 5: Setup UI layer and Uses-cases components

- Create all UI classes and their related xml layouts.
- Add related use cases in viewModel classes.



## Task 6: Setup dependency injection

- Add Dagger 2 components.
- Move all necessary dependencies to the Dagger 2 module.

## Task 7: Polish UI

- Use material components to ensure a unified design system across the app.
-