Money Matters: A Personal Finance Management App

HARI HARA SUDHAN T M

HEMANATHAN N G

RAKESH K

RAMAMOORTHI S

Introduction:

1.1 Overview

The app allows user to keep track of their expenses and accounts, and provides an overview of their financial status. users can set a budget for various expenses and view their progress towards it.

Project Workflow:

●Users register into the application.

●After registration and user logins into the application.

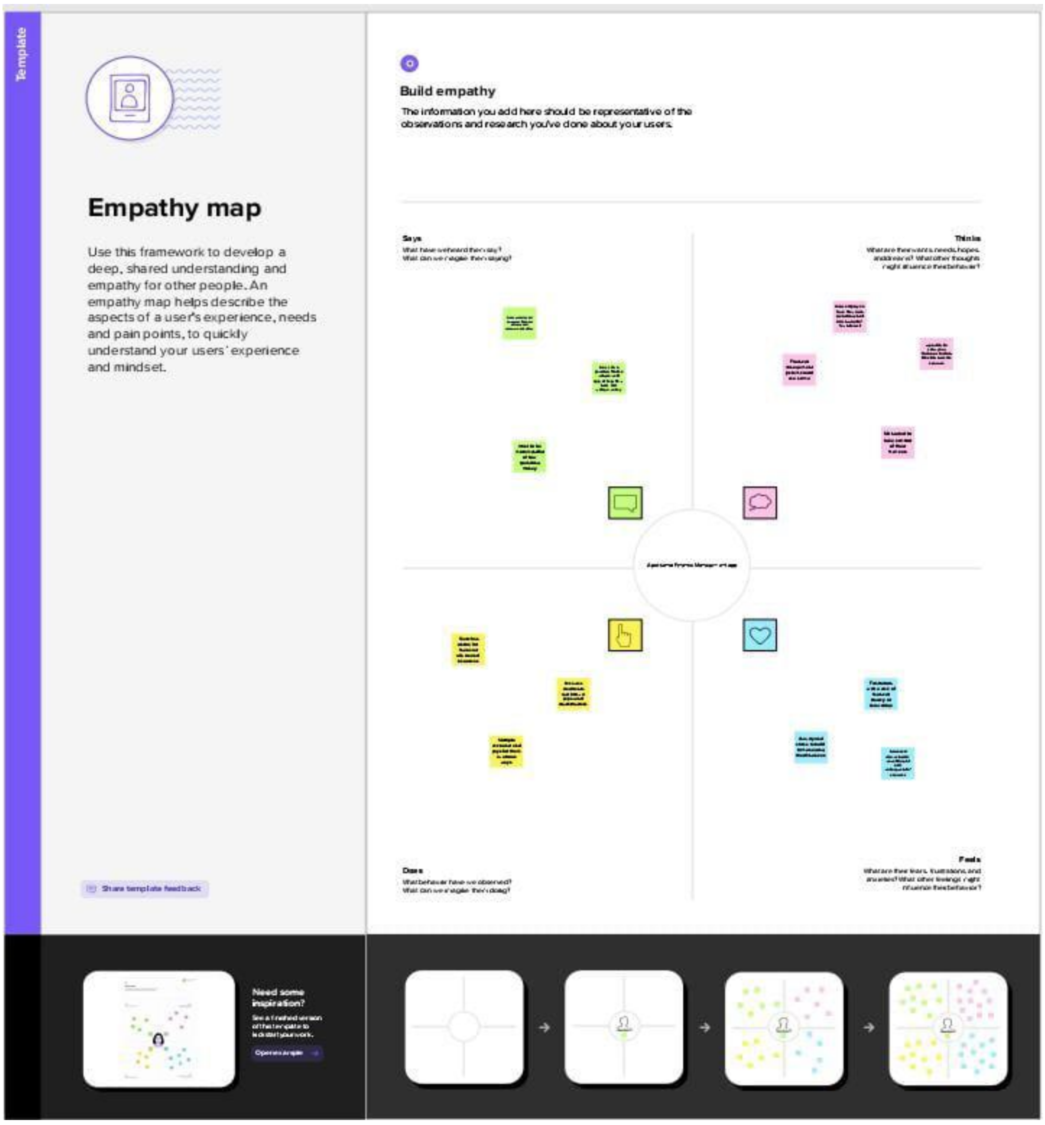
●User enters into the main page

1.2 Purpose

The purpose of an expense tracker is to help individuals and businesses keep track of their spending and manage their finances more effectively. An expense tracker typically involves recording and categorizing every expense, from bills and rent payments to grocery shopping and entertainment expenses.

2. Problem Definition and Design Thinking

2.1 Empathy Map



Template

Empathy map

Use this framework to develop a deep, shared understanding and empathy for other people. An empathy map helps describe the aspects of a user's experience, needs and pain points, to quickly understand your users' experience and mindset.

Share template feedback

Build empathy

The information you add here should be representative of the observations and research you've done about your users.

Says

What have we heard them say?
What can we imagine them saying?

Thinks

What are their wants, needs, hopes, and dreams? What other thoughts might influence their behavior?

Does

What behavior have we observed?
What can we imagine them doing?

Feels

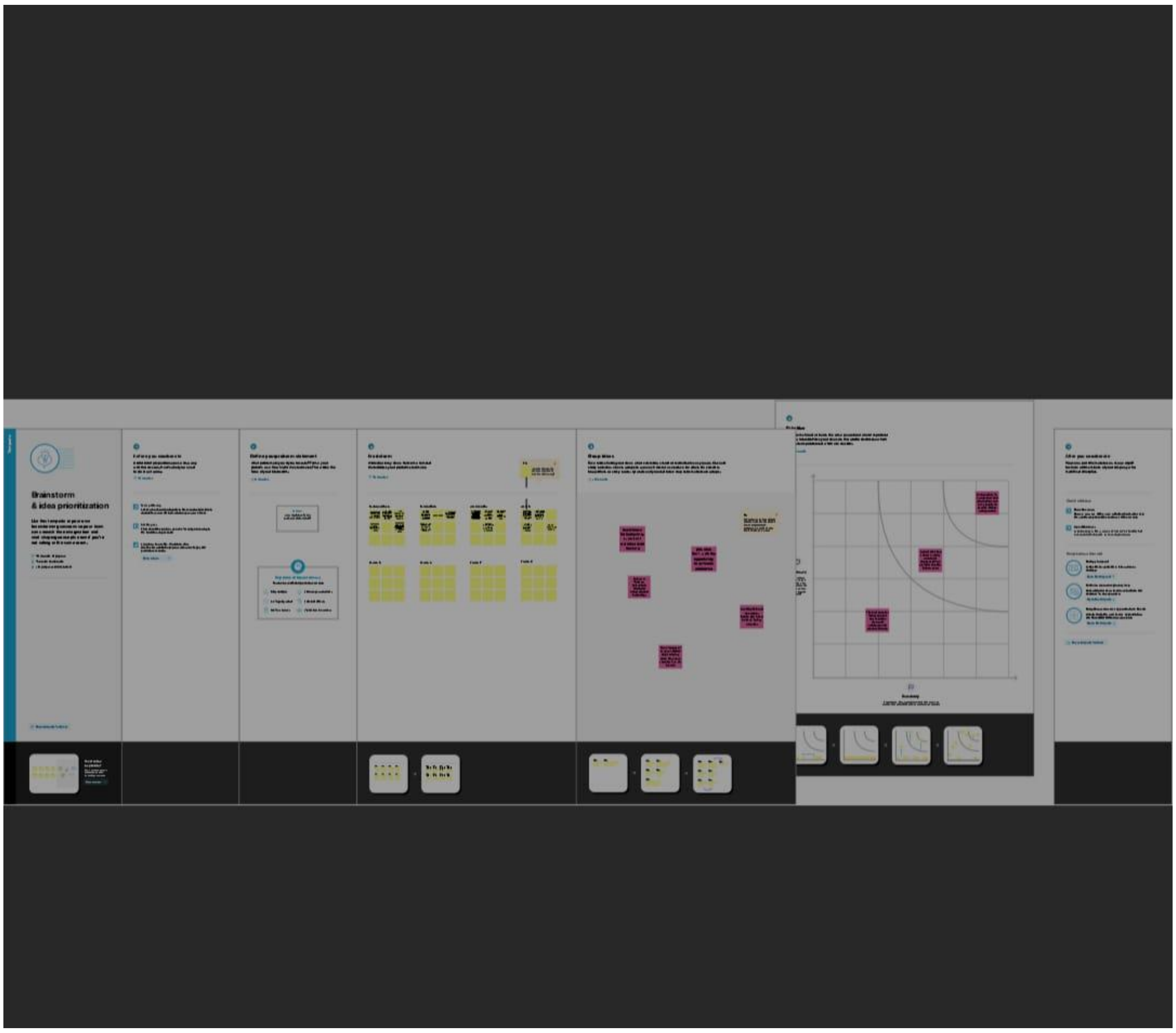
What are their fears, frustrations, and anxieties? What other feelings might influence their behavior?

Need some inspiration?

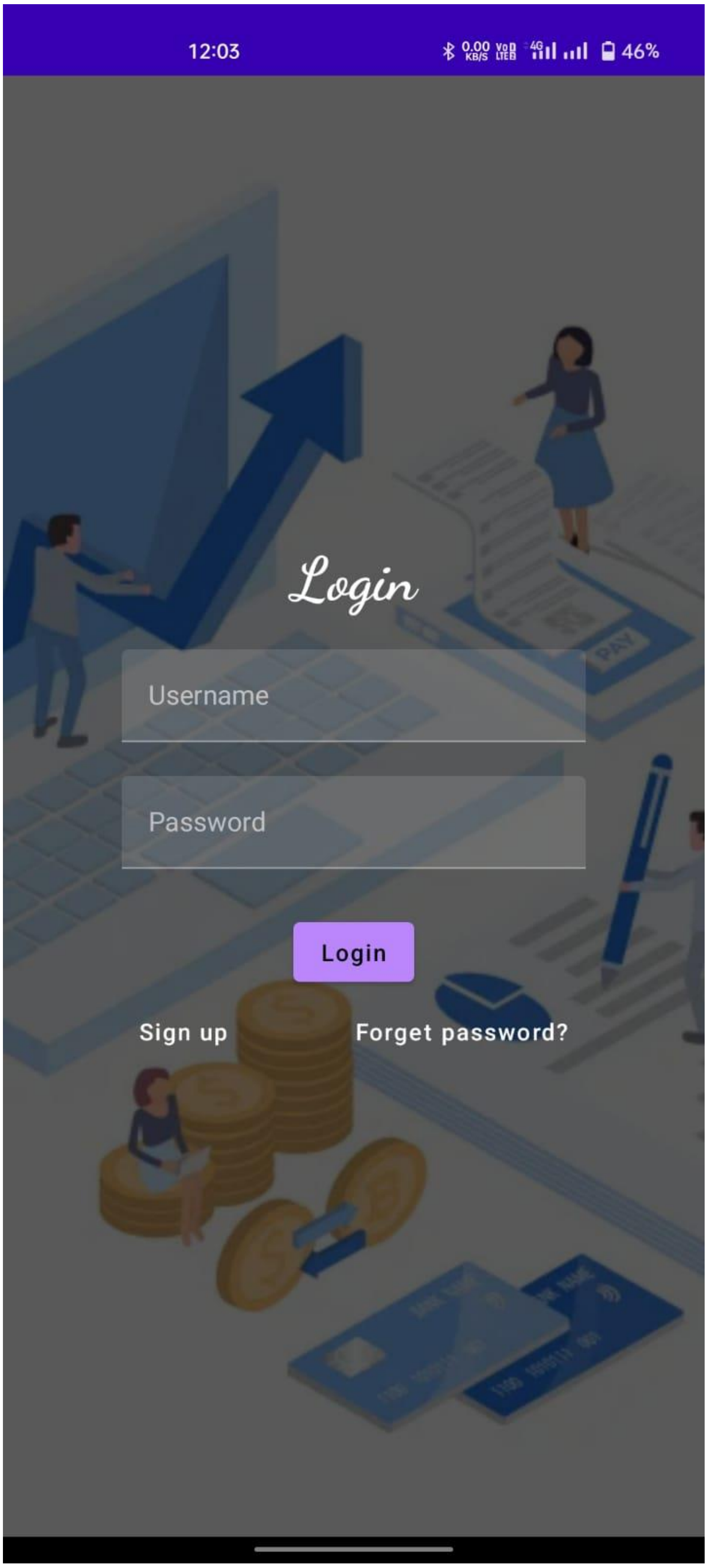
See a finished version of this template to kickstart your work.

Open example

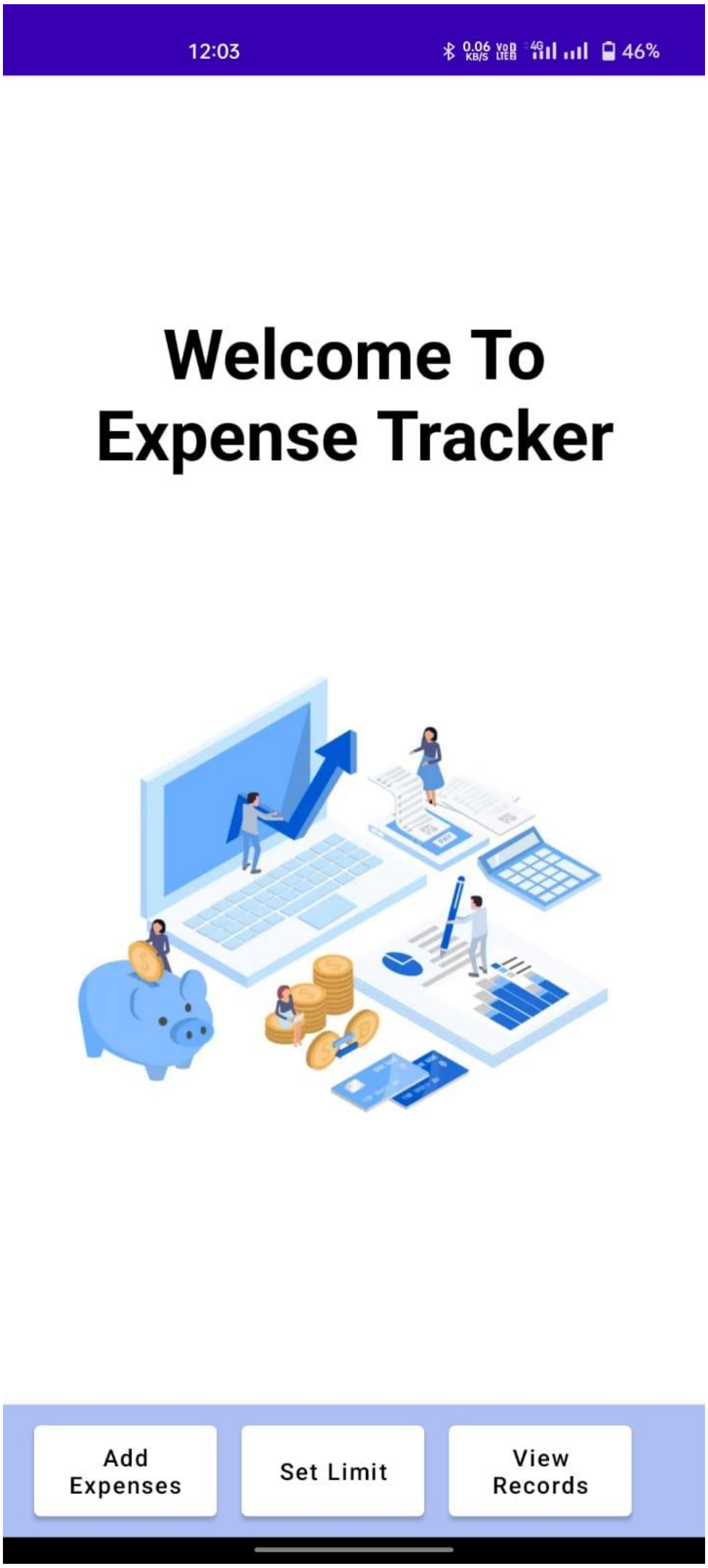
2.2 ideation and brainstorming



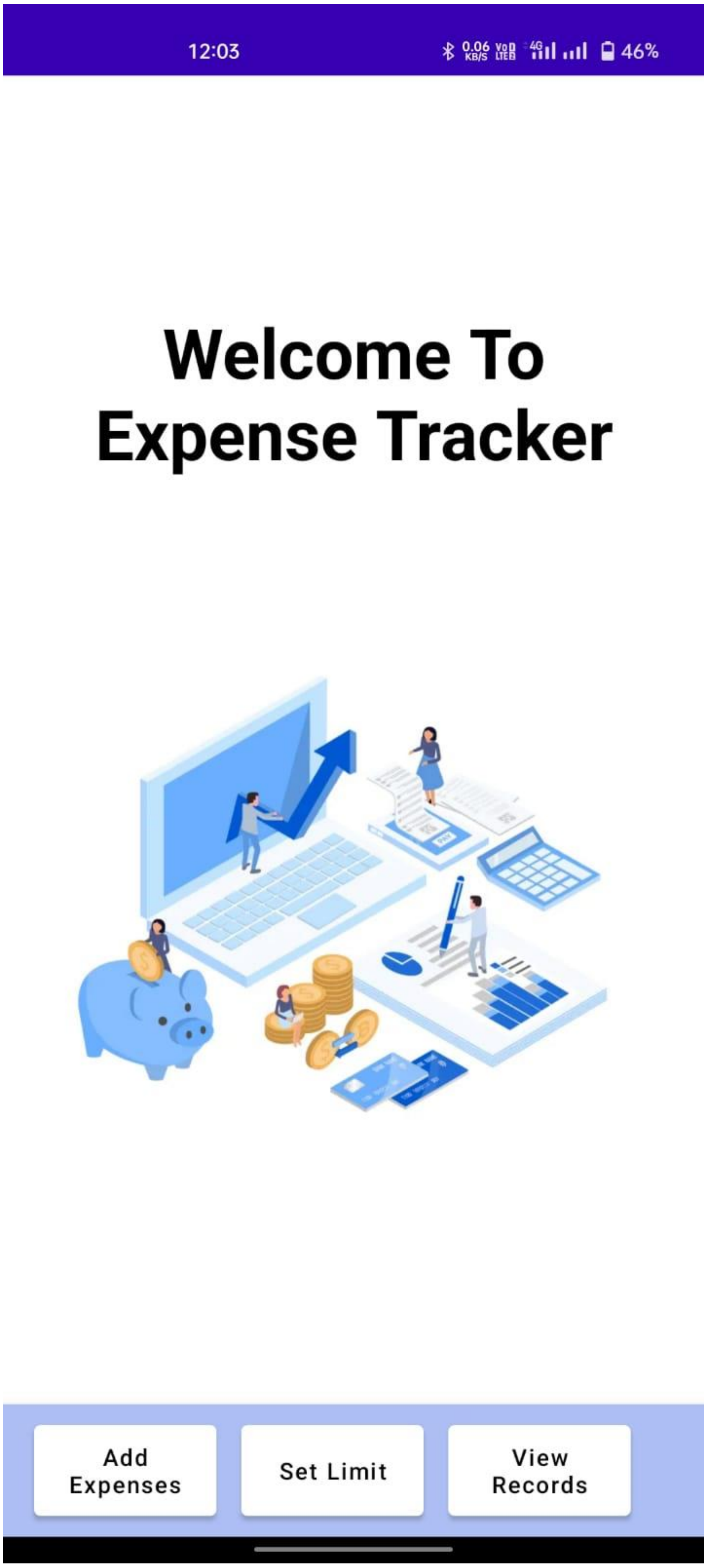
Login Page:


12:03
46%
Login
Username
Password
Login
Sign up
Forget password?

Register Page:



Main Podcast Page:



4.Advantages and Disadvantages

Advantages:

- Better financial awareness: An expense tracker provides a detailed record of all your expenses, giving you a better understanding of your spending habits and allowing you to make more informed financial decisions.

- Budgeting assistance: By categorizing your expenses and tracking your spending, an expense tracker can help you create and stick to a budget, ensuring that you don't overspend and have enough money for your needs.

- Improved tax preparation: By keeping track of all your expenses throughout the year, you can simplify the process of tax preparation and potentially save money on your taxes.

- Time-saving: Automated expense trackers can save you time by automatically categorizing expenses, eliminating the need for manual data entry.

Disadvantages:

- Time-consuming: Depending on the complexity of the expense tracker and the frequency with which you use it, it can be time-consuming to record every expense. This can be especially true if you need to manually enter each expense rather than using automated tools.

- Over-reliance: Relying too heavily on an expense tracker can create a sense of false security. It's important to remember that an expense tracker is only a tool and should not replace other important aspects of financial planning, such as setting a budget or saving for long-term goals.

- Technical issues: Like any software or app, expense trackers can sometimes experience technical difficulties that can result in lost data or inaccurate information. This can be frustrating and time-consuming to resolve

● Cost: While there are many free expense tracker apps available, some more advanced ones may come with a cost. Depending on your budget, this may be a drawback.

5.Application

Expense trackers can be used in a variety of personal and professional contexts to help individuals and organizations manage their finances more effectively. Here are a few examples of how expense trackers can be applied

Personal budgeting:

An expense tracker can help individuals create and stick to a personal budget by providing a clear picture of their spending habits and identifying areas where they can cut back.

Business expense tracking:

Businesses can use expense trackers to monitor employee spending, track business expenses, and simplify the process of filing taxes.

Tax preparation:

By tracking all expenses throughout the year, individuals can make tax preparation easier and more accurate, potentially resulting in lower tax bills or larger refunds.

Financial planning:

An expense tracker can help individuals and families plan for long-term financial goals by identifying areas where they can save money, invest more effectively, or reduce debt.

Project budgeting:

Organizations can use expense trackers to manage budgets for specific projects, ensuring that costs are kept under control and resources are allocated efficiently.

About Android Studio Application:

• Code editing: Android Studio provides a code editor with features like code completion, syntax highlighting, and refactoring. It supports multiple languages, including Java, Kotlin, and C++.

• Visual layout editor: Android Studio has a visual layout editor that allows developers to design the user interface (UI) of their app using a drag-and-drop interface. The layout editor provides tools to create complex layouts and preview how the UI will look on different devices.

• Build system: Android Studio includes Gradle, a build system used to compile and package Android apps. Gradle is highly customizable and allows developers to automate many aspects of the build process.

• Debugging and testing: Android Studio has tools for debugging and testing Android apps, including a debugger, emulator, and integration with various testing frameworks.

• Integration with Google services: Android Studio has built-in support for integrating Google services like Google Maps, Google Cloud Platform, and Firebase

About Android Studio

application:

♦ Studio is the official integrate development environment (IDE) for building Android Apps. It was first released by Google in 2013 and has since become the most popular development environment for Android app developers.

♦ Android Studio is based on the IntelliJ IDEA community edition, and it includes many tools and features designed specifically for developing Android apps Some of the key features of Android Studio include:

User Interface (UI) Designer:

Android Studio includes a powerful Designer that allows developers to easily create and modify app layouts using drag-and-drop tools. The UI designer supports a variety of layouts, including linear, relative, and constraint layouts.

Code Editor:

Android Studio includes a powerful code editor that provides syntax highlighting, code completion, and other features to help developers write clean, efficient code. The code editor also supports debugging and refactoring tools.

Emulator:

Android Studio includes a built-in emulator that allows developers to test their apps on different Android devices without needing to own the actual devices. The emulator supports a wide range of Android versions and device configurations.

Gradle Build System:

Android Studio uses the Gradle build system, which makes it easy to manage dependencies and build complex apps with multiple modules.

Version Control:

Android Studio supports version control systems like Git, allowing developers to easily manage their code changes and collaborate with other team members

Performance Profiling:

Android Studio includes performance profiling tools that allow developers to identify performance bottlenecks in their apps and optimize their code for better performance. Overall, Android Studio is a powerful tool for developing high-quality Android apps. It provides a range of features and tools that make it easy for developers to build, test, and deploy their apps.

6.Conclusion:

In conclusion, an expense tracker can be a valuable tool for individuals and businesses looking to manage their finances more effectively. By providing a detailed record of all expenses, an expense tracker can help you gain a better understanding of your spending habits, identify areas where you can cut back, and set financial goals for the future. Additionally, expense trackers can provide time-saving benefits, improve tax preparation, and help monitor your expenses for fraudulent or unauthorized transactions. However, it's important to keep in mind that expense trackers are just one tool in a larger financial planning toolkit, and they should not replace other important aspects of financial planning, such as budgeting and goal setting. Ultimately, the effectiveness of an expense tracker will depend on how regularly and accurately you use it, and the level of commitment you have towards managing your finances.

7.Future Scope

The future of expense trackers looks promising as the need for financial management continues to grow. Here are some potential areas of future growth and development for expense trackers:

Artificial intelligence and automation:

As technology advances, we can expect to see more expense trackers that use artificial intelligence and automation to categorize expenses and provide more accurate financial insights.

Integration with other financial tools:

In the future, we may see more expense trackers that integrate with other financial tools, such as investment management platforms or banking apps, to provide a more comprehensive view of an individual's finances.

Improved mobile capabilities:

With the increasing use of smartphones and mobile devices, we can expect to see more expense trackers that offer improved mobile capabilities, such as receipt scanning and real-time notifications.

Enhanced security:

As the risk of cyber threats and fraud increases, we can expect to see more expense trackers that offer enhanced security features, such as two-factor authentication and encryption.

Customizable features:

In the future, we may see more expense trackers that offer customizable features, allowing users to tailor the app to their specific needs and preferences.

Overall, the future of expense trackers is likely to be characterized by increased automation, integration with other financial tools, and improved security and customization features. As the need for financial management continues to grow, we can expect to see more individuals and businesses turn to expense trackers to help them manage their finances more effectively.

Appendix:

Build.gradle

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
}

android {
    namespace 'com.example.expensestracker'
    compileSdk 33

    defaultConfig {
        applicationId "com.example.expensestracker"
        minSdk 21
        targetSdk 33
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner
"androidx.test.runner.AndroidJUnitRunner"
        vectorDrawables {
            useSupportLibrary true
        }
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-
optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
    buildFeatures {
        compose true
    }
    composeOptions {
        kotlinCompilerExtensionVersion '1.2.0'
    }
    packagingOptions {
        resources {
            excludes += '/META-INF/{AL2.0,LGPL2.1}'
        }
    }
}

dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.3.1'
    implementation "androidx.compose.ui:ui:$compose_ui_version"
    implementation "androidx.compose.ui:ui-tooling-
preview:$compose_ui_version"

    implementation 'androidx.compose.material:material:1.2.0'

    implementation 'androidx.room:room-common:2.5.1'

    implementation 'androidx.room:room-ktx:2.5.1'

    testImplementation 'junit:junit:4.13.2'

    androidTestImplementation 'androidx.test.ext:junit:1.1.5'

    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'

    androidTestImplementation "androidx.compose.ui:ui-test-
junit4:$compose_ui_version"

    debugImplementation "androidx.compose.ui:ui-
tooling:$compose_ui_version"

    debugImplementation "androidx.compose.ui:ui-test-
manifest:$compose_ui_version"

}
```

LoginActivity.kt

```
package com.example.expensestracker


import android.content.Context

import android.content.Intent

import android.os.Bundle

import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

import androidx.compose.foundation.Image

import androidx.compose.foundation.layout.*

import androidx.compose.material.*

import androidx.compose.runtime.*

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color

import androidx.compose.ui.layout.ContentScale

import androidx.compose.ui.res.painterResource

import androidx.compose.ui.text.font.FontFamily

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.text.input.PasswordVisualTransformation

import androidx.compose.ui.text.input.VisualTransformation

import androidx.compose.ui.tooling.preview.Preview

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.core.content.ContextCompat

import com.example.expensestracker.ui.theme.ExpensesTrackerTheme


class LoginActivity : ComponentActivity() {

    private lateinit var databaseHelper: UserDatabaseHelper

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        databaseHelper = UserDatabaseHelper(this)

        setContent {

            ExpensesTrackerTheme {

                // A surface container using the 'background' color from the
theme

                Surface(

                    modifier = Modifier.fillMaxSize(),

                    color = MaterialTheme.colors.background

                ) {

                    LoginScreen(this, databaseHelper)

                }

            }

        }
    }
}
@Composable
fun LoginScreen(context: Context, databaseHelper: UserDatabaseHelper) {

    Image(
        painterResource(id = R.drawable.img_1), contentDescription = "",
        alpha =0.3F,
        contentScale = ContentScale.FillHeight,

        )

    var username by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }

    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {

        Text(
            fontSize = 36.sp,
            fontWeight = FontWeight.ExtraBold,
            fontFamily = FontFamily.Cursive,
            color = Color.White,
            text = "Login"
        )
        Spacer(modifier = Modifier.height(10.dp))
```

```
TextField(
    value = username,
    onValueChange = { username = it },
    label = { Text("Username") },
    modifier = Modifier.padding(10.dp)
        .width(280.dp)
)

TextField(
    value = password,
    onValueChange = { password = it },
    label = { Text("Password") },
    modifier = Modifier.padding(10.dp)
        .width(280.dp),
    visualTransformation = PasswordVisualTransformation()

)

if (error.isNotEmpty()) {
    Text(
        text = error,
        color = MaterialTheme.colors.error,
        modifier = Modifier.padding(vertical = 16.dp)
    )
}

Button(
    onClick = {
        if (username.isNotEmpty() && password.isNotEmpty()) {
            val user = databaseHelper.getUserByUsername(username)
            if (user != null && user.password == password) {
                error = "Successfully log in"
                context.startActivity(
                    Intent(
                        context,
                        MainActivity::class.java
                    )
                )
                //onLoginSuccess()
            }
            else {
                error =  "Invalid username or password"
            }

        } else {
            error = "Please fill all fields"
        }
    },
    modifier = Modifier.padding(top = 16.dp)
) {
    Text(text = "Login")
}
Row {
    TextButton(onClick = {context.startActivity(
        Intent(
            context,
            RegisterActivity::class.java
        )
    )}
```

```
            )
            { Text(color = Color.White,text = "Sign up") }
            TextButton(onClick = {
            })

            {
                Spacer(modifier = Modifier.width(60.dp))
                Text(color = Color.White,text = "Forget password?")
            }
        }
    }
}
private fun startMainPage(context: Context) {
    val intent = Intent(context, MainActivity::class.java)
    ContextCompat.startActivity(context, intent, null)
}
```

MainActivity.kt

```
package com.example.expensestracker

import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.text.input.VisualTransformation
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.core.content.ContextCompat
import com.example.expensestracker.ui.theme.ExpensesTrackerTheme


class LoginActivity : ComponentActivity() {
    private lateinit var databaseHelper: UserDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = UserDatabaseHelper(this)
        setContent {
            ExpensesTrackerTheme {
                // A surface container using the 'background' color from the
theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colors.background
                ) {
                    LoginScreen(this, databaseHelper)
                }
            }
        }
    }
}
@Composable
fun LoginScreen(context: Context, databaseHelper: UserDatabaseHelper) {

    Image(
        painterResource(id = R.drawable.img_1), contentDescription = "",
        alpha =0.3F,
        contentScale = ContentScale.FillHeight,

        )

    var username by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }

    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {

        Text(
            fontSize = 36.sp,
            fontWeight = FontWeight.ExtraBold,
            fontFamily = FontFamily.Cursive,
            color = Color.White,
            text = "Login"
        )
        Spacer(modifier = Modifier.height(10.dp))

        TextField(
            value = username,
            onValueChange = { username = it },
            label = { Text("Username") },
            modifier = Modifier.padding(10.dp)
                .width(280.dp)
        )

        TextField(
            value = password,
            onValueChange = { password = it },
            label = { Text("Password") },
            modifier = Modifier.padding(10.dp)
                .width(280.dp),
            visualTransformation = PasswordVisualTransformation()

        )

        if (error.isNotEmpty()) {
            Text(
                text = error,
                color = MaterialTheme.colors.error,
                modifier = Modifier.padding(vertical = 16.dp)
            )
        }
```

```
Button(
    onClick = {
        if (username.isNotEmpty() && password.isNotEmpty()) {
            val user = databaseHelper.getUserByUsername(username)
            if (user != null && user.password == password) {
                error = "Successfully log in"
                context.startActivity(
                    Intent(
                        context,
                        MainActivity::class.java
                    )
                )
                //onLoginSuccess()
            }
            else {
                error =  "Invalid username or password"
            }


        } else {
            error = "Please fill all fields"
        }
    },
    modifier = Modifier.padding(top = 16.dp)
) {
    Text(text = "Login")
}
Row {
    TextButton(onClick = {context.startActivity(
        Intent(
            context,
                    RegisterActivity::class.java
                )
            )}
            )
            { Text(color = Color.White,text = "Sign up") }
            TextButton(onClick = {
            })

            {
                Spacer(modifier = Modifier.width(60.dp))
                Text(color = Color.White,text = "Forget password?")
            }
        }
    }
}
private fun startMainPage(context: Context) {
    val intent = Intent(context, MainActivity::class.java)
    ContextCompat.startActivity(context, intent, null)
}
```

RegisterActivity.kt

```
package com.example.expensestracker

import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image

import androidx.compose.foundation.layout.*

import androidx.compose.material.*

import androidx.compose.runtime.*

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.layout.ContentScale

import androidx.compose.ui.res.painterResource

import androidx.compose.ui.text.font.FontFamily

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.text.input.PasswordVisualTransformation

import androidx.compose.ui.tooling.preview.Preview

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.core.content.ContextCompat

import com.example.expensestracker.ui.theme.ExpensesTrackerTheme


class RegisterActivity : ComponentActivity() {

    private lateinit var databaseHelper: UserDatabaseHelper

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        databaseHelper = UserDatabaseHelper(this)

        setContent {

            ExpensesTrackerTheme {

                // A surface container using the 'background' color from the
theme

                Surface(

                    modifier = Modifier.fillMaxSize(),

                    color = MaterialTheme.colors.background
                ) {

                    RegistrationScreen(this,databaseHelper)
                }
            }
        }
    }
}

@Composable
fun RegistrationScreen(context: Context, databaseHelper:
UserDatabaseHelper) {

    Image(
        painterResource(id = R.drawable.img_1), contentDescription = "",
        alpha =0.3F,
        contentScale = ContentScale.FillHeight,

        )

    var username by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var email by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }

    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
```

```
Text(
    fontSize = 36.sp,
    fontWeight = FontWeight.ExtraBold,
    fontFamily = FontFamily.Cursive,
    color = Color.White,
    text = "Register"
)

Spacer(modifier = Modifier.height(10.dp))
TextField(
    value = username,
    onValueChange = { username = it },
    label = { Text("Username") },
    modifier = Modifier
        .padding(10.dp)
        .width(280.dp)


)

TextField(
    value = email,
    onValueChange = { email = it },
    label = { Text("Email") },
    modifier = Modifier
        .padding(10.dp)
        .width(280.dp)
)

TextField(
            value = password,
            onValueChange = { password = it },
            label = { Text("Password") },
            modifier = Modifier
                .padding(10.dp)
                .width(280.dp),
            visualTransformation = PasswordVisualTransformation()
        )



        if (error.isNotEmpty()) {
            Text(
                text = error,
                color = MaterialTheme.colors.error,
                modifier = Modifier.padding(vertical = 16.dp)
            )
        }

        Button(
            onClick = {
                if (username.isNotEmpty() && password.isNotEmpty() &&
email.isNotEmpty()) {
                    val user = User(
                        id = null,
                        firstName = username,
                        lastName = null,
                        email = email,
                        password = password
                    )
                    databaseHelper.insertUser(user)
                    error = "User registered successfully"
                    // Start LoginActivity using the current context
                    context.startActivity(
                        Intent(
                            context,
                            LoginActivity::class.java
                        )
                    )

                } else {
                    error = "Please fill all fields"
                }
            },
            modifier = Modifier.padding(top = 16.dp)
        ) {
            Text(text = "Register")
        }
        Spacer(modifier = Modifier.width(10.dp))
        Spacer(modifier = Modifier.height(10.dp))

        Row() {
            Text(
                modifier = Modifier.padding(top = 14.dp), text = "Have an
account?"
            )
            TextButton(onClick = {
                context.startActivity(
                    Intent(
                        context,
                        LoginActivity::class.java
                    )
                )
            })

            {
                Spacer(modifier = Modifier.width(10.dp))
                Text(text = "Log in")
            }
        }
      }
}
private fun startLoginActivity(context: Context) {
        val intent = Intent(context, LoginActivity::class.java)
        ContextCompat.startActivity(context, intent, null)
}
```

SetLimitActivity.kt

```
package com.example.expensestracker

import android.annotation.SuppressLint
import android.content.Context
import android.content.Intent
import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.LazyRow
import androidx.compose.foundation.lazy.items
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.example.expensestracker.ui.theme.ExpensesTrackerTheme

class SetLimitActivity : ComponentActivity() {
    private lateinit var expenseDatabaseHelper: ExpenseDatabaseHelper
    @SuppressLint("UnusedMaterialScaffoldPaddingParameter")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        expenseDatabaseHelper = ExpenseDatabaseHelper(this)
        setContent {
            Scaffold(
                // in scaffold we are specifying top bar.
                bottomBar = {
                    // inside top bar we are specifying
                    // background color.
                    BottomAppBar(backgroundColor = Color(0xFFadbef4),
            modifier = Modifier.height(80.dp),
            // along with that we are specifying
            // title for our top bar.
            content = {

                Spacer(modifier = Modifier.width(15.dp))

                Button(
                    onClick = {
                        startActivity(
                            Intent(
                                applicationContext,
                                AddExpensesActivity::class.java
                            )
                        )
                    },
                    colors = ButtonDefaults.buttonColors(backgroundColor
= Color.White),
                    modifier = Modifier.size(height = 55.dp, width = 110.dp)
                )
                {
                    Text(
                        text = "Add Expenses", color = Color.Black, fontSize
= 14.sp,
                        textAlign = TextAlign.Center
                    )
                }

                Spacer(modifier = Modifier.width(15.dp))
```

```
                        Button(
                            onClick = {
                                startActivity(
                                    Intent(
                                        applicationContext,
                                        SetLimitActivity::class.java
                                    )
                                )
                            },
                            colors = ButtonDefaults.buttonColors(backgroundColor
= Color.White),
                            modifier = Modifier.size(height = 55.dp, width = 110.dp)
                        )
                        {
                            Text(
                                text = "Set Limit", color = Color.Black, fontSize =
14.sp,
                                textAlign = TextAlign.Center
                            )
                        }

                        Spacer(modifier = Modifier.width(15.dp))

                        Button(
                            onClick = {
                                startActivity(
                                    Intent(
                                        applicationContext,
                                        ViewRecordsActivity::class.java
                                    )
                                )
                            },
                            colors = ButtonDefaults.buttonColors(backgroundColor
= Color.White),
                            modifier = Modifier.size(height = 55.dp, width = 110.dp)
                        )
                        {
                            Text(
                                text = "View Records", color = Color.Black, fontSize
= 14.sp,
                                textAlign = TextAlign.Center
                            )
                        }

                    }
                )
            }
        ) {
            val data=expenseDatabaseHelper.getAllExpense();
            Log.d("swathi" ,data.toString())
            val expense = expenseDatabaseHelper.getAllExpense()
            Limit(this, expenseDatabaseHelper,expense)
        }
    }
  }
}
```

```
@Composable
fun Limit(context: Context, expenseDatabaseHelper:
ExpenseDatabaseHelper, expense: List<Expense>) {
    Column(
        modifier = Modifier
            .padding(top = 100.dp, start = 30.dp)
            .fillMaxHeight()
            .fillMaxWidth(),
        horizontalAlignment = Alignment.Start
    ) {

        var amount by remember { mutableStateOf("") }
        var error by remember { mutableStateOf("") }

        Text(text = "Monthly Amount Limit", fontWeight = FontWeight.Bold,
fontSize = 20.sp)
        Spacer(modifier = Modifier.height(10.dp))
        TextField(value = amount, onValueChange = { amount = it },
            label = { Text(text = "Set Amount Limit ") })

        Spacer(modifier = Modifier.height(20.dp))

        if (error.isNotEmpty()) {
            Text(
                text = error,
                color = MaterialTheme.colors.error,
                modifier = Modifier.padding(vertical = 16.dp)
            )
        }
```

```
Button(onClick = {
    if (amount.isNotEmpty()) {
        val expense = Expense(
            id = null,
            amount = amount
        )
        expenseDatabaseHelper.insertExpense(expense)
    }
}) {
    Text(text = "Set Limit")
}

Spacer(modifier = Modifier.height(10.dp))

LazyRow(
    modifier = Modifier
        .fillMaxSize()
        .padding(top = 0.dp),

    horizontalArrangement = Arrangement.Start
) {
    item {

        LazyColumn {
            items(expense) { expense ->
                Column(
                        ) {
                            Text("Remaining Amount: ${expense.amount}",
fontWeight = FontWeight.Bold)
                        }
                    }
                }
            }

        }
    }
}



//@Composable
//fun Records(expense: List<Expense>) {
//    Text(text = "View Records", modifier = Modifier.padding(top = 24.dp,
start = 106.dp, bottom = 24.dp ), fontSize = 30.sp)
//    Spacer(modifier = Modifier.height(30.dp))
//    LazyRow(
//        modifier = Modifier
//            .fillMaxSize()
//            .padding(top = 80.dp),
//
//        horizontalArrangement = Arrangement.SpaceBetween
//    ){
//        item {
//
//            LazyColumn {
//                items(expense) { expense ->
//                    Column(modifier = Modifier.padding(top = 16.dp, start =
48.dp, bottom = 20.dp)) {
//                        Text("Remaining Amount: ${expense.amount}")
//                    }
//                }
//            }
//        }
//
//    }
//}
```

User

```
package com.example.expensestracker

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "user_table")
data class User(
    @PrimaryKey(autoGenerate = true) val id: Int?,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?,
    @ColumnInfo(name = "email") val email: String?,
    @ColumnInfo(name = "password") val password: String?,
    )
```

UserDao

```
package com.example.expensestracker

import androidx.room.*

@Dao
interface UserDao {

    @Query("SELECT * FROM user_table WHERE email = :email")
    suspend fun getUserByEmail(email: String): User?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertUser(user: User)

    @Update
    suspend fun updateUser(user: User)

    @Delete
    suspend fun deleteUser(user: User)
}
```

UserDatabase

```
package com.example.expensestracker

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(entities = [User::class], version = 1)
abstract class UserDatabase : RoomDatabase() {

    abstract fun userDao(): UserDao

    companion object {

        @Volatile
        private var instance: UserDatabase? = null

        fun getDatabase(context: Context): UserDatabase {
            return instance ?: synchronized(this) {
                val newInstance = Room.databaseBuilder(
                    context.applicationContext,
                    UserDatabase::class.java,
                    "user_database"
                ).build()
                instance = newInstance
                newInstance
            }
        }
    }
}
```

UserDatabaseHelper

```
package com.example.expensestracker

import android.annotation.SuppressLint
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

class UserDatabaseHelper(context: Context) :
    SQLiteOpenHelper(context, DATABASE_NAME, null,
DATABASE_VERSION) {

    companion object {
        private const val DATABASE_VERSION = 1
        private const val DATABASE_NAME = "UserDatabase.db"

        private const val TABLE_NAME = "user_table"
        private const val COLUMN_ID = "id"
        private const val COLUMN_FIRST_NAME = "first_name"
        private const val COLUMN_LAST_NAME = "last_name"
        private const val COLUMN_EMAIL = "email"

        private const val COLUMN_PASSWORD = "password"

    }


    override fun onCreate(db: SQLiteDatabase?) {

        val createTable = "CREATE TABLE $TABLE_NAME (" +

                "$COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT, " +

                "$COLUMN_FIRST_NAME TEXT, " +

                "$COLUMN_LAST_NAME TEXT, " +

                "$COLUMN_EMAIL TEXT, " +

                "$COLUMN_PASSWORD TEXT" +

                ")"


        db?.execSQL(createTable)

    }


    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int,
newVersion: Int) {

        db?.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")

        onCreate(db)

    }


    fun insertUser(user: User) {

        val db = writableDatabase

        val values = ContentValues()

        values.put(COLUMN_FIRST_NAME, user.firstName)

        values.put(COLUMN_LAST_NAME, user.lastName)

        values.put(COLUMN_EMAIL, user.email)
        values.put(COLUMN_PASSWORD, user.password)

        db.insert(TABLE_NAME, null, values)

        db.close()

    }


    @SuppressLint("Range")

    fun getUserByUsername(username: String): User? {

        val db = readableDatabase

        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME
WHERE $COLUMN_FIRST_NAME = ?", arrayOf(username))

        var user: User? = null

        if (cursor.moveToFirst()) {

            user = User(

                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),

                firstName =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),

                lastName =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),

                email =
cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),

                password =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),

            )

        }

        cursor.close()

        db.close()

        return user

    }

    @SuppressLint("Range")

    fun getUserById(id: Int): User? {
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME
WHERE $COLUMN_ID = ?", arrayOf(id.toString()))
        var user: User? = null
        if (cursor.moveToFirst()) {
            user = User(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                firstName =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                lastName =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                email =
cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                password =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
            )
        }
        cursor.close()
        db.close()
        return user
    }

    @SuppressLint("Range")
    fun getAllUsers(): List<User> {
        val users = mutableListOf<User>()
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME",
null)
        if (cursor.moveToFirst()) {
            do {
                val user = User(
                    id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                    firstName =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                    lastName =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                    email =
cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                    password =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
                )
                users.add(user)
            } while (cursor.moveToNext())
        }
        cursor.close()
        db.close()
        return users
    }

}
```

Items

```
package com.example.expensestracker

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey
```

```
@Entity(tableName = "items_table")
data class Items(
    @PrimaryKey(autoGenerate = true) val id: Int?,
    @ColumnInfo(name = "item_name") val itemName: String?,
    @ColumnInfo(name = "quantity") val quantity: String?,
    @ColumnInfo(name = "cost") val cost: String?,
)
```

ItemsDao

```
package com.example.expensestracker

import androidx.room.*

@Dao
interface ItemsDao {

    @Query("SELECT * FROM items_table WHERE  cost= :cost")
    suspend fun getItemsByCost(cost: String): Items?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertItems(items: Items)

    @Update
    suspend fun updateItems(items: Items)

    @Delete
    suspend fun deleteItems(items: Items)
}
```

ItemDatabase.kt

```
package com.example.expensestracker

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(entities = [Items::class], version = 1)
abstract class ItemsDatabase : RoomDatabase() {

    abstract fun ItemsDao(): ItemsDao

    companion object {

        @Volatile
        private var instance: ItemsDatabase? = null

        fun getDatabase(context: Context): ItemsDatabase {
            return instance ?: synchronized(this) {
                val newInstance = Room.databaseBuilder(
                    context.applicationContext,
                    ItemsDatabase::class.java,
                    "items_database"
                ).build()
                instance = newInstance
                newInstance
            }
        }
    }
}
```

ItemsDatabaseHelper

```
package com.example.expensestracker

import android.annotation.SuppressLint
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper


class ItemsDatabaseHelper(context: Context) :
    SQLiteOpenHelper(context, DATABASE_NAME,
null,DATABASE_VERSION){

    companion object {
        private const val DATABASE_VERSION = 1
        private const val DATABASE_NAME = "ItemsDatabase.db"
        private const val TABLE_NAME = "items_table"

        private const val COLUMN_ID = "id"

        private const val COLUMN_ITEM_NAME = "item_name"

        private const val COLUMN_QUANTITY = "quantity"

        private const val COLUMN_COST = "cost"

    }


    override fun onCreate(db: SQLiteDatabase?) {

        val createTable = "CREATE TABLE $TABLE_NAME (" +

                "${COLUMN_ID} INTEGER PRIMARY KEY AUTOINCREMENT, " +

                "${COLUMN_ITEM_NAME} TEXT," +

                "${COLUMN_QUANTITY} TEXT," +

                "${COLUMN_COST} TEXT" +

                ")"


        db?.execSQL(createTable)

    }


    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int,
newVersion: Int) {

        db?.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")

        onCreate(db)

    }


    fun insertItems(items: Items) {

        val db = writableDatabase

        val values = ContentValues()
        values.put(COLUMN_ITEM_NAME, items.itemName)

        values.put(COLUMN_QUANTITY, items.quantity)

        values.put(COLUMN_COST, items.cost)

        db.insert(TABLE_NAME, null, values)

        db.close()

    }




    @SuppressLint("Range")

    fun getItemsByCost(cost: String): Items? {

        val db = readableDatabase

        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME
WHERE $COLUMN_COST = ?", arrayOf(cost))

        var items: Items? = null

        if (cursor.moveToFirst()) {

            items = Items(

                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),

                itemName =
cursor.getString(cursor.getColumnIndex(COLUMN_ITEM_NAME)),

                quantity =
cursor.getString(cursor.getColumnIndex(COLUMN_QUANTITY)),

                cost = cursor.getString(cursor.getColumnIndex(COLUMN_COST)),

            )

        }

        cursor.close()

        db.close()

        return items

    }
    @SuppressLint("Range")
    fun getItemsById(id: Int): Items? {
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME
WHERE $COLUMN_ID = ?", arrayOf(id.toString()))
        var items: Items? = null
        if (cursor.moveToFirst()) {
            items = Items(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                itemName =
cursor.getString(cursor.getColumnIndex(COLUMN_ITEM_NAME)),
                quantity =
cursor.getString(cursor.getColumnIndex(COLUMN_QUANTITY)),
                cost = cursor.getString(cursor.getColumnIndex(COLUMN_COST)),
            )
        }
        cursor.close()
        db.close()
        return items
    }

    @SuppressLint("Range")
    fun getAllItems(): List<Items> {
        val item = mutableListOf<Items>()
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME",
null)
        if (cursor.moveToFirst()) {
            do {
                val items = Items(
                    id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                    itemName = 
cursor.getString(cursor.getColumnIndex(COLUMN_ITEM_NAME)),
                    quantity = 
cursor.getString(cursor.getColumnIndex(COLUMN_QUANTITY)),
                    cost = 
cursor.getString(cursor.getColumnIndex(COLUMN_COST)),
                )
                item.add(items)
            } while (cursor.moveToNext())
        }
        cursor.close()
        db.close()
        return item
    }


}
```

Expense

```
package com.example.expensestracker

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey
```

```
@Entity(tableName = "expense_table")
data class Expense(
    @PrimaryKey(autoGenerate = true) val id: Int?,
    @ColumnInfo(name = "amount") val amount: String?,
)
```

ExpenseDao

```
package com.example.expensestracker

import androidx.room.*

@Dao
interface ExpenseDao {

    @Query("SELECT * FROM expense_table WHERE  amount= :amount")
    suspend fun getExpenseByAmount(amount: String): Expense?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertExpense(items: Expense)

    @Update
    suspend fun updateExpense(items: Expense)

    @Delete
    suspend fun deleteExpense(items: Expense)
}
```

ExpenseDatabase

```
package com.example.expensestracker

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(entities = [Items::class], version = 1)
abstract class ExpenseDatabase : RoomDatabase() {

    abstract fun ExpenseDao(): ItemsDao

    companion object {

        @Volatile
        private var instance: ExpenseDatabase? = null

        fun getDatabase(context: Context): ExpenseDatabase {
            return instance ?: synchronized(this) {
                val newInstance = Room.databaseBuilder(
                    context.applicationContext,
                    ExpenseDatabase::class.java,
                    "expense_database"
                ).build()
                instance = newInstance
                newInstance
            }
        }
    }
}
```

ExpenseDatabaseHelper.kt

```
package com.example.expensestracker

import android.annotation.SuppressLint
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

class ExpenseDatabaseHelper(context: Context) :
    SQLiteOpenHelper(context, DATABASE_NAME,
null,DATABASE_VERSION){

    companion object {
        private const val DATABASE_VERSION = 1
        private const val DATABASE_NAME = "ExpenseDatabase.db"

        private const val TABLE_NAME = "expense_table"
        private const val COLUMN_ID = "id"
        private const val COLUMN_AMOUNT = "amount"
    }

    override fun onCreate(db: SQLiteDatabase?) {
        val createTable = "CREATE TABLE $TABLE_NAME (" +
                "${COLUMN_ID} INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "${COLUMN_AMOUNT} TEXT" +
                ")"

        db?.execSQL(createTable)
    }

    override fun onUpgrade(db1: SQLiteDatabase?, oldVersion: Int,
newVersion: Int) {
        db1?.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
        onCreate(db1)
    }

    fun insertExpense(expense: Expense) {
        val db1 = writableDatabase
        val values = ContentValues()
        values.put(COLUMN_AMOUNT, expense.amount)
        db1.insert(TABLE_NAME, null, values)
        db1.close()
    }

    fun updateExpense(expense: Expense) {
        val db = writableDatabase
        val values = ContentValues()
        values.put(COLUMN_AMOUNT, expense.amount)
        db.update(TABLE_NAME, values, "$COLUMN_ID=?", 
arrayOf(expense.id.toString()))
        db.close()
    }




    @SuppressLint("Range")
    fun getExpenseByAmount(amount: String): Expense? {
        val db1 = readableDatabase
        val cursor: Cursor = db1.rawQuery("SELECT * FROM 
${ExpenseDatabaseHelper.TABLE_NAME} WHERE 
${ExpenseDatabaseHelper.COLUMN_AMOUNT} = ?", arrayOf(amount))
        var expense: Expense? = null
        if (cursor.moveToFirst()) {
            expense = Expense(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                amount = 
cursor.getString(cursor.getColumnIndex(COLUMN_AMOUNT)),
            )
        }
        cursor.close()
        db1.close()
        return expense
    }
    @SuppressLint("Range")
    fun getExpenseById(id: Int): Expense? {
        val db1 = readableDatabase
        val cursor: Cursor = db1.rawQuery("SELECT * FROM $TABLE_NAME
WHERE $COLUMN_ID = ?", arrayOf(id.toString()))

        var expense: Expense? = null

        if (cursor.moveToFirst()) {

            expense = Expense(

                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),

                amount =
cursor.getString(cursor.getColumnIndex(COLUMN_AMOUNT)),

            )

        }

        cursor.close()

        db1.close()

        return expense

    }

    @SuppressLint("Range")

    fun getExpenseAmount(id: Int): Int? {

        val db = readableDatabase

        val query = "SELECT $COLUMN_AMOUNT FROM $TABLE_NAME
WHERE $COLUMN_ID=?"

        val cursor = db.rawQuery(query, arrayOf(id.toString()))

        var amount: Int? = null

        if (cursor.moveToFirst()) {

            amount =
cursor.getInt(cursor.getColumnIndex(COLUMN_AMOUNT))

        }

        cursor.close()

        db.close()

        return amount

    }
    @SuppressLint("Range")
    fun getAllExpense(): List<Expense> {
        val expenses = mutableListOf<Expense>()
        val db1 = readableDatabase
        val cursor: Cursor = db1.rawQuery("SELECT * FROM $TABLE_NAME",
null)
        if (cursor.moveToFirst()) {
            do {
                val expense = Expense(
                    id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                    amount =
cursor.getString(cursor.getColumnIndex(COLUMN_AMOUNT)),
                )
                expenses.add(expense)
            } while (cursor.moveToNext())
        }
        cursor.close()
        db1.close()
        return expenses
    }






}
```

AddExpenseActivity.kt

```
package com.example.expensestracker
```

```kotlin
import android.annotation.SuppressLint

import android.content.Context

import android.content.Intent

import android.os.Bundle

import android.widget.Toast

import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

import androidx.compose.foundation.layout.*

import androidx.compose.material.*

import androidx.compose.runtime.*

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.platform.LocalContext

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.text.style.TextAlign

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp


class AddExpensesActivity : ComponentActivity() {

    private lateinit var itemsDatabaseHelper: ItemsDatabaseHelper

    private lateinit var expenseDatabaseHelper: ExpenseDatabaseHelper

    @SuppressLint("UnusedMaterialScaffoldPaddingParameter")

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        itemsDatabaseHelper = ItemsDatabaseHelper(this)

        expenseDatabaseHelper = ExpenseDatabaseHelper(this)
        setContent {

            Scaffold(

                // in scaffold we are specifying top bar.

                bottomBar = {

                    // inside top bar we are specifying

                    // background color.

                    BottomAppBar(backgroundColor = Color(0xFFadbef4),

                        modifier = Modifier.height(80.dp),

                        // along with that we are specifying

                        // title for our top bar.

                        content = {


                            Spacer(modifier = Modifier.width(15.dp))


                            Button(

                                onClick =
{startActivity(Intent(applicationContext,AddExpensesActivity::class.java))}
,

                                colors = ButtonDefaults.buttonColors(backgroundColor
= Color.White),

                                modifier = Modifier.size(height = 55.dp, width = 110.dp)

                            )

                            {

                                Text(

                                    text = "Add Expenses", color = Color.Black, fontSize
= 14.sp,

                                    textAlign = TextAlign.Center

                                )

                            }
```

```
              Spacer(modifier = Modifier.width(15.dp))

              Button(
                  onClick = {
                      startActivity(
                          Intent(
                              applicationContext,
                              SetLimitActivity::class.java
                          )
                      )
                  },
                  colors = ButtonDefaults.buttonColors(backgroundColor
= Color.White),
                  modifier = Modifier.size(height = 55.dp, width = 110.dp)
              )
              {
                  Text(
                      text = "Set Limit", color = Color.Black, fontSize =
14.sp,
                      textAlign = TextAlign.Center
                  )
              }

              Spacer(modifier = Modifier.width(15.dp))

              Button(
                  onClick = {
                      startActivity(
                                        Intent(
                                            applicationContext,
                                            ViewRecordsActivity::class.java
                                        )
                                    )
                                },
                                colors = ButtonDefaults.buttonColors(backgroundColor
= Color.White),
                                modifier = Modifier.size(height = 55.dp, width = 110.dp)
                            )
                            {
                                Text(
                                    text = "View Records", color = Color.Black, fontSize
= 14.sp,
                                    textAlign = TextAlign.Center
                                )
                            }

                        }
                    )
                }
            ) {
                AddExpenses(this, itemsDatabaseHelper,
expenseDatabaseHelper)
            }
        }
    }
}
```

```
@SuppressLint("Range")
@Composable
fun AddExpenses(context: Context, itemsDatabaseHelper:
ItemsDatabaseHelper, expenseDatabaseHelper: ExpenseDatabaseHelper)
{
    Column(
        modifier = Modifier
            .padding(top = 100.dp, start = 30.dp)
            .fillMaxHeight()
            .fillMaxWidth(),
        horizontalAlignment = Alignment.Start
    ) {

        val mContext = LocalContext.current
        var items by remember { mutableStateOf("") }
        var quantity by remember { mutableStateOf("") }
        var cost by remember { mutableStateOf("") }
        var error by remember { mutableStateOf("") }

        Text(text = "Item Name", fontWeight = FontWeight.Bold, fontSize =
20.sp)
        Spacer(modifier = Modifier.height(10.dp))
        TextField(value = items, onValueChange = { items = it },
            label = { Text(text = "Item Name") })

        Spacer(modifier = Modifier.height(20.dp))
        Text(text = "Quantity of item", fontWeight = FontWeight.Bold, fontSize
= 20.sp)
        Spacer(modifier = Modifier.height(10.dp))
        TextField(value = quantity, onValueChange = { quantity = it },
            label = { Text(text = "Quantity") })

        Spacer(modifier = Modifier.height(20.dp))

        Text(text = "Cost of the item", fontWeight = FontWeight.Bold, fontSize
= 20.sp)
        Spacer(modifier = Modifier.height(10.dp))
        TextField(value = cost, onValueChange = { cost = it },
            label = { Text(text = "Cost") })

        Spacer(modifier = Modifier.height(20.dp))

        if (error.isNotEmpty()) {
            Text(
                text = error,
                color = MaterialTheme.colors.error,
                modifier = Modifier.padding(vertical = 16.dp)
            )
        }

        Button(onClick = {
            if (items.isNotEmpty() && quantity.isNotEmpty() &&
cost.isNotEmpty()) {
                val items = Items(
                    id = null,
                    itemName = items,
                    quantity = quantity,
                    cost = cost
                )


                val limit= expenseDatabaseHelper.getExpenseAmount(1)



                val actualvalue = limit?.minus(cost.toInt())
               // Toast.makeText(mContext, actualvalue.toString(), 
Toast.LENGTH_SHORT).show()

                val expense = Expense(
                    id = 1,
                    amount = actualvalue.toString()
                )
                if (actualvalue != null) {
                    if (actualvalue < 1) {
                        Toast.makeText(mContext, "Limit Over", 
Toast.LENGTH_SHORT).show()
                    } else  {
                        expenseDatabaseHelper.updateExpense(expense)
                        itemsDatabaseHelper.insertItems(items)
                    }
                }

            }
        }) {
            Text(text = "Submit")
        }

    }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.ExpensesTracker"
        tools:targetApi="31">
        <activity
            android:name=".RegisterActivity"
            android:exported="false"
            android:label="@string/title_activity_register"
            android:theme="@style/Theme.ExpensesTracker" />
```

```
<activity
    android:name=".MainActivity"
    android:exported="false"
    android:label="MainActivity"
    android:theme="@style/Theme.ExpensesTracker" />
<activity
    android:name=".ViewRecordsActivity"
    android:exported="false"
    android:label="@string/title_activity_view_records"
    android:theme="@style/Theme.ExpensesTracker" />
<activity
    android:name=".SetLimitActivity"
    android:exported="false"
    android:label="@string/title_activity_set_limit"
    android:theme="@style/Theme.ExpensesTracker" />
<activity
    android:name=".AddExpensesActivity"
    android:exported="false"
    android:label="@string/title_activity_add_expenses"
    android:theme="@style/Theme.ExpensesTracker" />
<activity
    android:name=".LoginActivity"
    android:exported="true"
    android:label="@string/app_name"
    android:theme="@style/Theme.ExpensesTracker">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
```

```
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```