

Money Matters:

A Personal Finance Management App

Introduction

1.1 Overview

Managing personal finances can be a challenge for many people, especially those with busy schedules and multiple financial obligations. To help individuals track their expenses, set budgets, and manage their money effectively, you have developed "Money Matters," a personal finance management app.

With "Money Matters," users can easily monitor their income, expenses, and investments, set financial goals, and generate reports to gain insight into their spending habits. The app provides a user-friendly interface and a range of features that enable users to stay on top of their finances and make informed decisions about their money.

Whether users are looking to save for a specific goal, reduce debt, or simply gain better control of their finances, "Money Matters" offers a comprehensive solution that helps them achieve their financial objectives.

1.2 Purpose

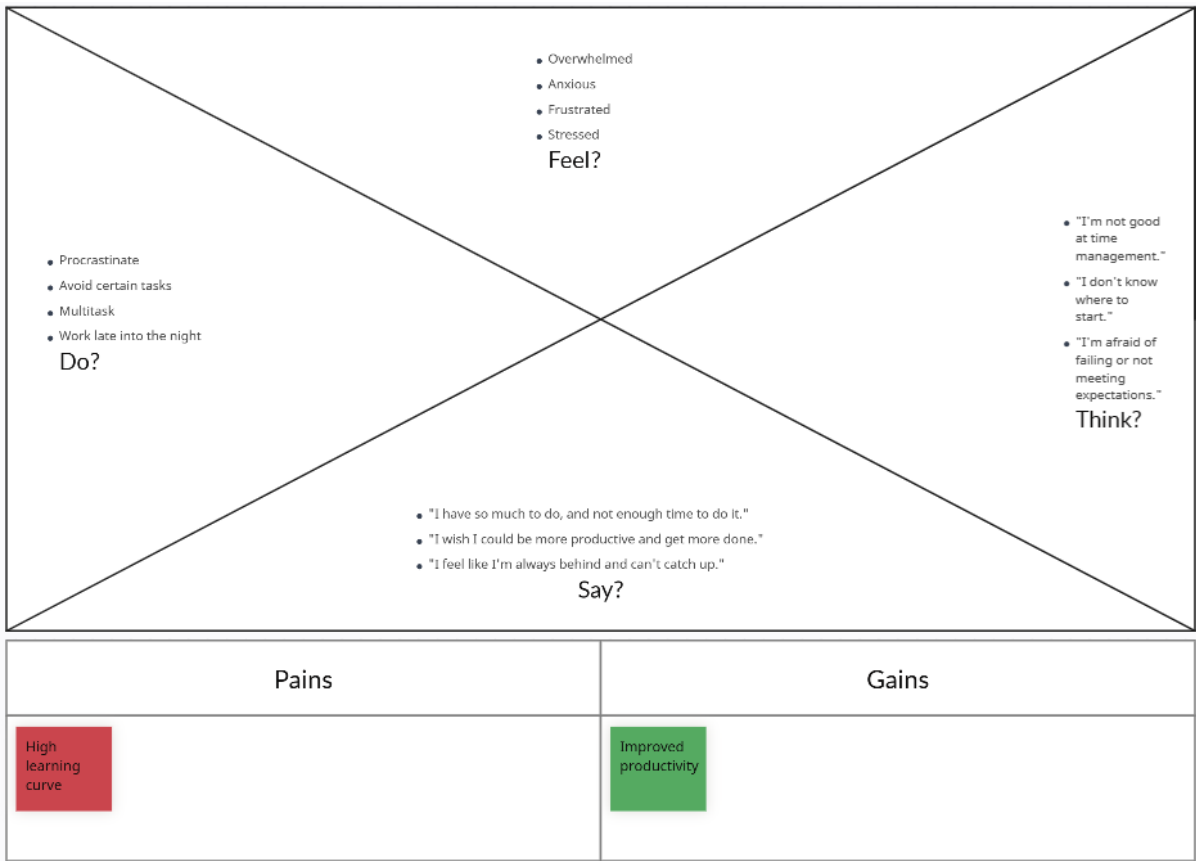
The purpose of this project is to develop a personal finance management app that helps individuals keep track of their financial activities, manage their money effectively, and achieve their financial goals. This app will enable users to track their expenses, create budgets, monitor investments, and generate reports on their financial status. By using this app, individuals can gain a better understanding of their financial situation and take control of their finances. The app will provide users with real-time updates on their account balances, transaction history, and investment performance, allowing them to make informed financial decisions. Ultimately, this app aims to empower individuals to manage their money with ease, save more, and achieve financial stability.

The benefits of using a personal finance management app are numerous. First and foremost, it helps users to stay organized and keep track of their financial activities in one place. With the ability to monitor transactions and expenses in real-time, individuals can identify areas where they may be overspending and make necessary adjustments to their budget. The app also provides insights into spending patterns and trends, enabling users to make informed decisions about their money.

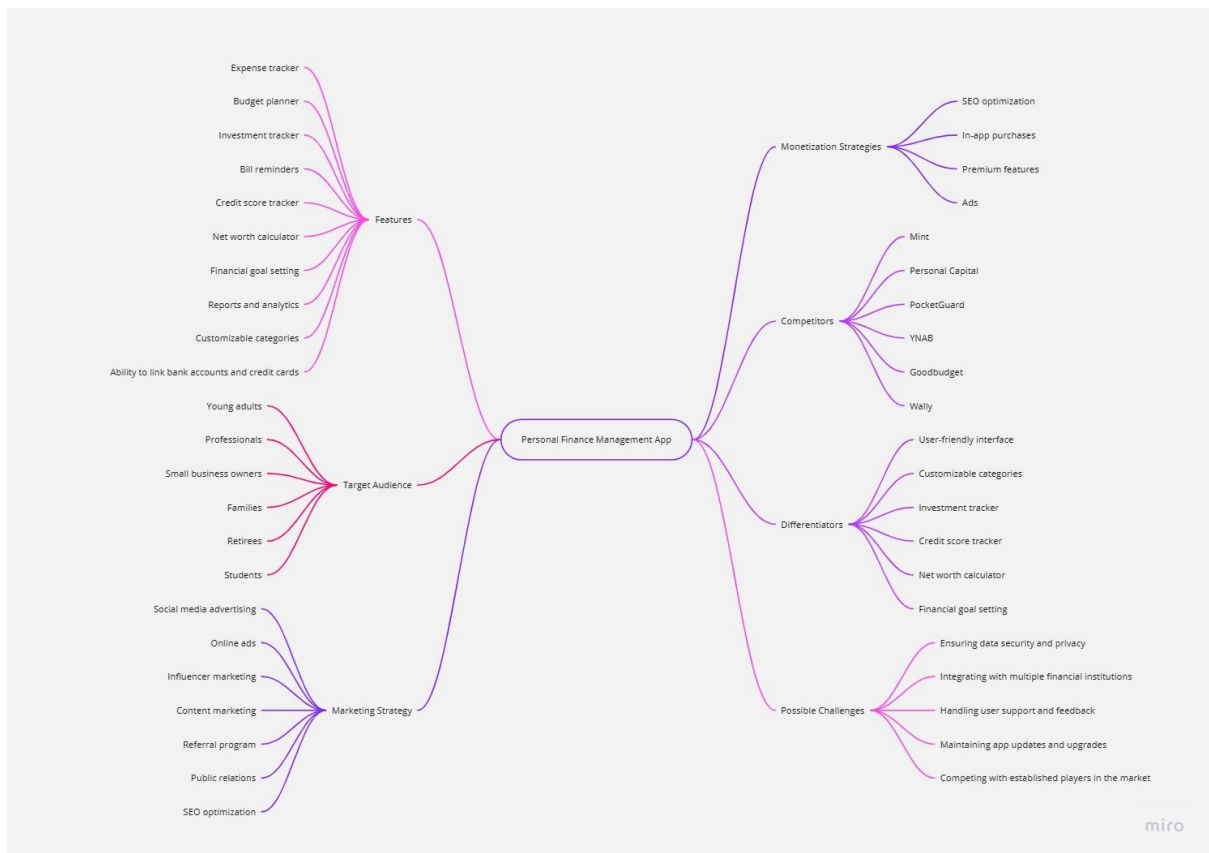
In addition, the app can be customized to meet individual needs and preferences. Users can set their financial goals and track their progress towards achieving them. They can also customize alerts and notifications to stay on top of bill payments, account balances, and investment performance. Moreover, the app can be synced with bank accounts and credit cards, making it easy to monitor and manage multiple accounts in one place.

Problem Definition & Design Thinking

2.1 Empathy Map

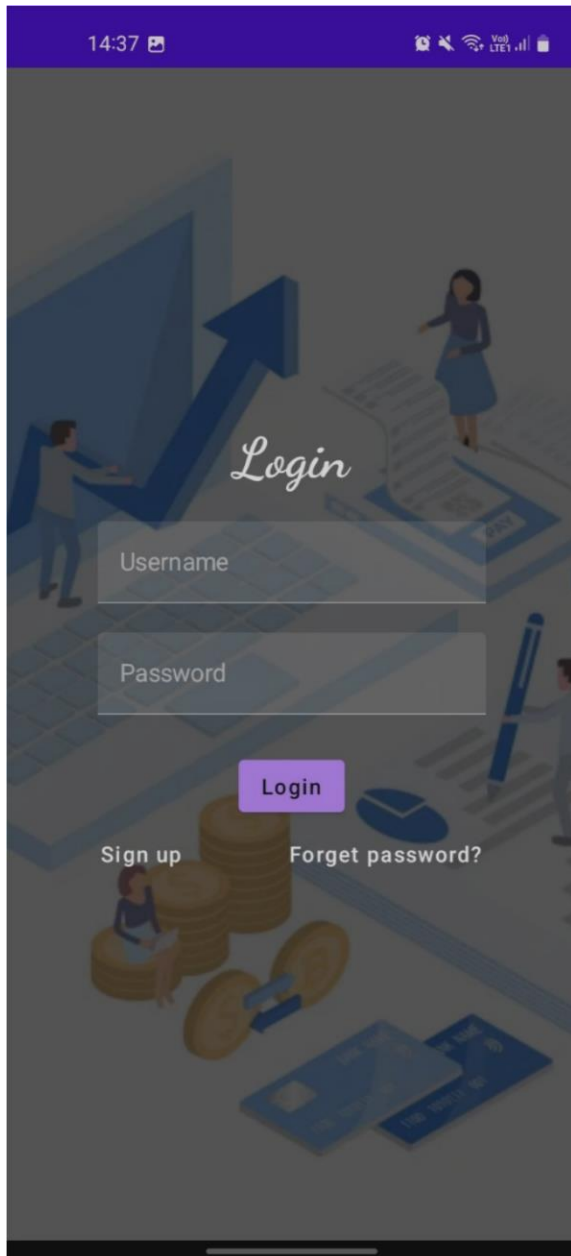


2.2 Ideation & Brainstroming Map

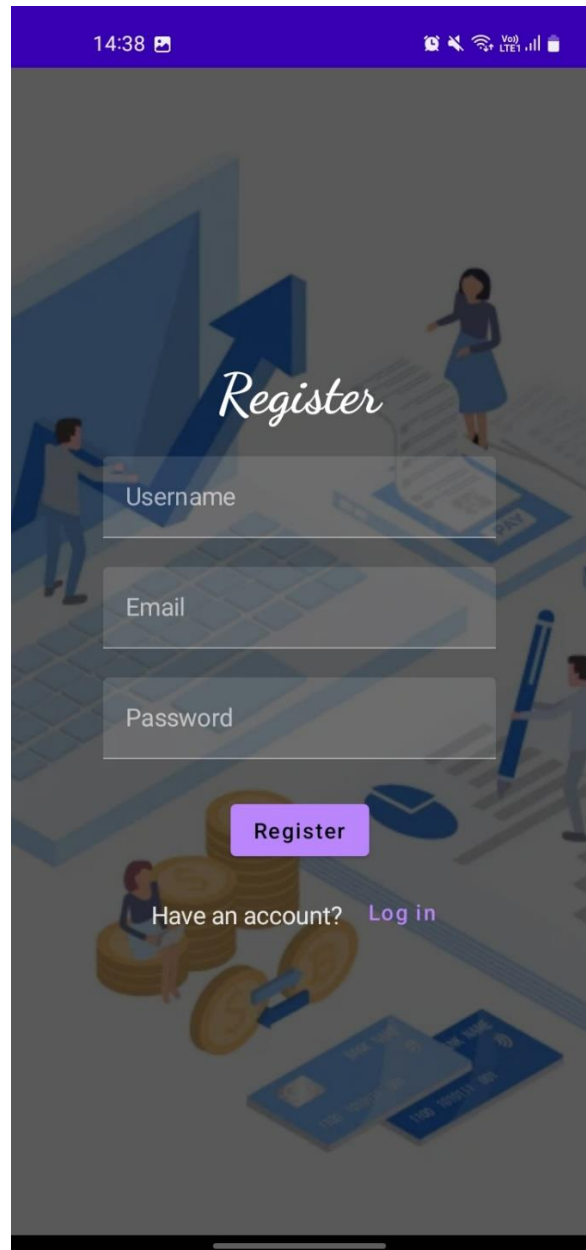


Result :

LOGIN PAGE :



REGISTER PAGE :



MAIN PAGE :



ADD EXPENSES PAGE



Item Name

Item Name
pizza

Quantity of item

Quantity
2







Cost of the item

Cost
400

Submit

SET LIMIT PAGE BEFORE ADDING ANY DATA IN EXPENSES:

14:39



Monthly Amount Limit

Set Limit

Remaining Amount: 10000







Add Expenses

Set Limit

View Records

VIEW RECORDS PAGE:

14:39



View Records

Item_Name: pizza
Quantity: 2
Cost: 400

Item_Name: cake
Quantity: 3
Cost: 300

Add Expenses

Set Limit

View Records

SET LIMIT PAGE AFTER ADDING EXPENSE IN ADD EXPENSE PAGE:



Monthly Amount Limit

Set Amount Limit

Set Limit

Remaining Amount: 9300



Advantages & Disadvantages

Advantages:

- **Convenience:** A personal finance management app provides users with the convenience of tracking their finances on their smartphones or other devices at any time and from anywhere.
- **Budgeting:** The app can help users create and stick to a budget, which can lead to better financial management and improved savings.
- **Automation:** The app can automate tasks such as bill payments, which can help users avoid missed payments and late fees.
- **Customization:** Personal finance management apps can be customized to suit the user's specific financial goals and needs.
- **Data Analysis:** The app can provide users with insights and analysis on their spending patterns, which can help them make informed financial decisions.

Disadvantages:

- **Cost:** Developing a personal finance management app can be costly, especially if you need to hire developers and designers.
- **Security:** Personal finance management apps handle sensitive financial information, so security is a major concern. Developers need to ensure that the app is secure and user data is protected.
- **User Adoption:** Getting users to adopt a new app can be a challenge, especially if they are already using a similar app or are not interested in financial management.
- **Technical issues:** Developing an app can involve technical issues such as bugs, compatibility issues, and updates, which can impact the user experience.
- **Data reliability:** The accuracy of financial data collected by the app depends on the user's input, so there is a risk of incorrect or incomplete data affecting the app's analysis and recommendations.

Applications

- **Personal Finance:** The primary area where the app can be applied is personal finance. The app can help individuals manage their money, track expenses, set budgets, and achieve their financial goals.
- **Small Business Management:** Small business owners can use the app to track expenses, create invoices, and manage their cash flow. The app can help business owners stay on top of their finances and make informed decisions about their business.
- **Education:** The app can be used as an educational tool to teach students about financial literacy. The app can provide interactive lessons and quizzes to help students learn about budgeting, saving, and investing.
- **Financial Planning:** Financial planners can use the app to help their clients manage their money and achieve their financial goals. The app can provide financial planners with real-time data about their clients' finances, making it easier to create personalized financial plans.
- **Investment Management:** The app can be used by investors to track their investments, monitor their portfolio, and make informed investment decisions. The app can provide investors with real-time data about the stock market, as well as insights into emerging trends and opportunities.
- **Banking:** Banks can use the app to provide their customers with a digital banking experience. The app can allow customers to check their account balances, transfer funds, and pay bills from their mobile devices.

Conclusion

In conclusion, developing a personal finance management app is a great way to help people take control of their finances and manage their money effectively. Through the process of designing and building the app, you have learned about the importance of defining the purpose and scope of the app, researching existing apps, choosing a development platform, designing the user interface, developing the app, adding features and functionalities, launching and promoting the app, and collecting feedback to make improvements.

By creating a well-designed and user-friendly app, you have the potential to make a positive impact on people's financial lives. With the increasing importance of financial literacy and the need for effective personal finance management, your app has the potential to be a valuable tool for many individuals and households.

Keep in mind that building an app is an ongoing process and there will always be room for improvement. As you continue to gather feedback and make improvements, your app will become even more useful and effective in helping people manage their money. Congratulations on completing your project and best of luck in your future endeavors!

Future Scope

- **Adding new features:** As technology and financial practices evolve, there will be new features you can add to the app. For example, you could integrate the app with digital wallets, cryptocurrency exchanges, or investment platforms.
- **Customization:** Users often want to customize their apps to their own preferences. You could explore adding customization options, such as different themes or color schemes, to make the app more personalized.
- **Artificial Intelligence:** You could explore integrating artificial intelligence (AI) into the app to provide users with personalized financial advice or help them make investment decisions. AI can also be used to automate routine financial tasks, such as paying bills or transferring funds.
- **Data Analytics:** As more data is collected through the app, you could explore using data analytics to provide insights into user behavior, spending patterns, and financial trends. This data could be used to improve the app's features and functionalities or to develop new services.
- **Collaboration:** You could explore partnering with financial institutions or other companies to offer additional services through the app. For example, you could partner with a bank to offer loan or credit card applications through the app.
- **International Expansion:** As the app gains popularity, you could explore expanding it to new markets around the world. This would require adapting the app to different languages, currencies, and financial regulations.

Appendix

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"
        />
    </application>
</manifest>
```

```

        <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>

```

AddExpensesActivity.kt

```

package com.example.expensetracker

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

    @SuppressWarnings("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,AddExpensesActivi
ty::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")
    }
}
}
}

```

Expense.kt

```
package com.example.expensetracker
```

```
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.kt

```
package com.example.expensetracker
```

```
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.kt

```

package com.example.expensetracker

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,

```

ExpenseDatabaseHelper.kt

```

    }

    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
    }
}

```

```

}

```

Items.kt

```

package com.example.expensetracker

```

```

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.kt

```

package com.example.expensetracker

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)
}

```

ItemsDatabase.kt

```

package com.example.expensetracker

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.kt

```

package com.example.expensetracker

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()
    }

    @SuppressWarnings("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 getItemById(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
}

}

```

LoginActivity.kt

```
package com.example.expensetracker
```

```

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

    TextField(
        value = password,
        onChange = { 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.expensetracker

import android.annotation.SuppressLint
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.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import
com.example.expensetracker.ui.theme.ExpensesTrackerTheme

```

```

class MainActivity : ComponentActivity() {

@SuppressLint("UnusedMaterialScaffoldPaddingParameter")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        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,AddExpensesActivi
ty::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
                            )
                        }
                    }
                }
            )
        }
    }
}

```

```

Modifier.width(15.dp))

        Spacer(modifier =

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

            )
        )
    )
}

```



```
}  
}
```

RegisterActivity.kt

```
package com.example.expensetracker  
  
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.expensetracker.ui.theme.ExpensesTrackerTheme  
  
class RegisterActivity : ComponentActivity() {  
    private lateinit var dbHelper:  
        UserDataHelper  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        dbHelper = UserDataHelper(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,
        onChange = { username = it },
        label = { Text("Username") },
        modifier = Modifier
            .padding(10.dp)
            .width(280.dp)
    )

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

    TextField(
        value = password,
        onChange = { 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.expensetracker

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

```

[illegible]

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.kt

```
package com.example.expensetracker
```

```
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.kt

```
package com.example.expensetracker  
  
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.kt

```
package com.example.expensetracker  
  
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.kt

```

package com.example.expensetracker

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()
    }

    @SuppressWarnings("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
    }
    @SuppressWarnings("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
    }

    @SuppressWarnings("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
    }
}

```

ViewRecordsActivity.kt

```
package com.example.expensetracker
```

```
import android.annotation.SuppressLint
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.ScrollState
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.foundation.verticalScroll
import androidx.compose.material.*
import androidx.compose.runtime.Composable
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.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import
com.example.expensetracker.ui.theme.ExpensesTrackerTheme

class ViewRecordsActivity : ComponentActivity() {
    private lateinit var itemsDatabaseHelper:
ItemsDatabaseHelper

@SuppressLint("UnusedMaterialScaffoldPaddingParameter",
"SuspiciousIndentation")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        itemsDatabaseHelper = ItemsDatabaseHelper(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=itemsDatabaseHelper.getAllItems();
                        Log.d("swathi" ,data.toString())
                        val items =
itemsDatabaseHelper.getAllItems()
                        Records(items)
                    }
                }
            }
        }
    }
}

```

```
@Composable
fun Records(items: List<Items>) {
    Text(text = "View Records", modifier =
        Modifier.padding(top = 24.dp, start = 106.dp, bottom =
            24.dp ), fontSize = 30.sp, fontWeight = FontWeight.Bold)
    Spacer(modifier = Modifier.height(30.dp))
    LazyRow(
        modifier = Modifier
            .fillMaxSize()
            .padding(top = 80.dp),

        horizontalArrangement = Arrangement.SpaceBetween
    ){
        item {

            LazyColumn {
                items(items) { items ->
                    Column(modifier = Modifier.padding(top
                        = 16.dp, start = 48.dp, bottom = 20.dp)) {
                        Text("Item_Name:
                            ${items.itemName}")
                    }
                }
            }
        }
    }
}
```

```
    ${items.quantity}")
```

```
    Text("Quantity:
```

```
    Text("Cost: ${items.cost}")
```

```
    }
```

```
    }
```

```
    }
```

```
    }
```

```
    }
```

```
}
```