Project Report

Introduction:

A personal finance app is created with the purpose of managing private money, tracking the spendings, and plan out a budget. Often these apps are also called Money Saving apps, as they enable the app users to borrow, lend, and even invest money. Now the target audience of the personal finance apps is the end-users, hence their business model is B2C. These apps help people wisely manage their money, and they are for the personal use of people and not to establish business goals.

A report from AppsFlyer states that these days an increasing number of people are installing and deploying finance apps on their smartphones. In 2019, it was registered that the users' activity on these apps had increased by 354% as compared to the year 2014. This kind of popularity explains the many benefits these finance apps bring to their users.

Purpose:

Efficiently Organize Finances & Monthly Budget

The personal finance app allows us to keep track of how much money is spent on certain purchases, analyzing incomes & expenses, and making overall smart decisions. It also helps in evaluating financial behavior and making the necessary adjustments.

Securely Manage Finance

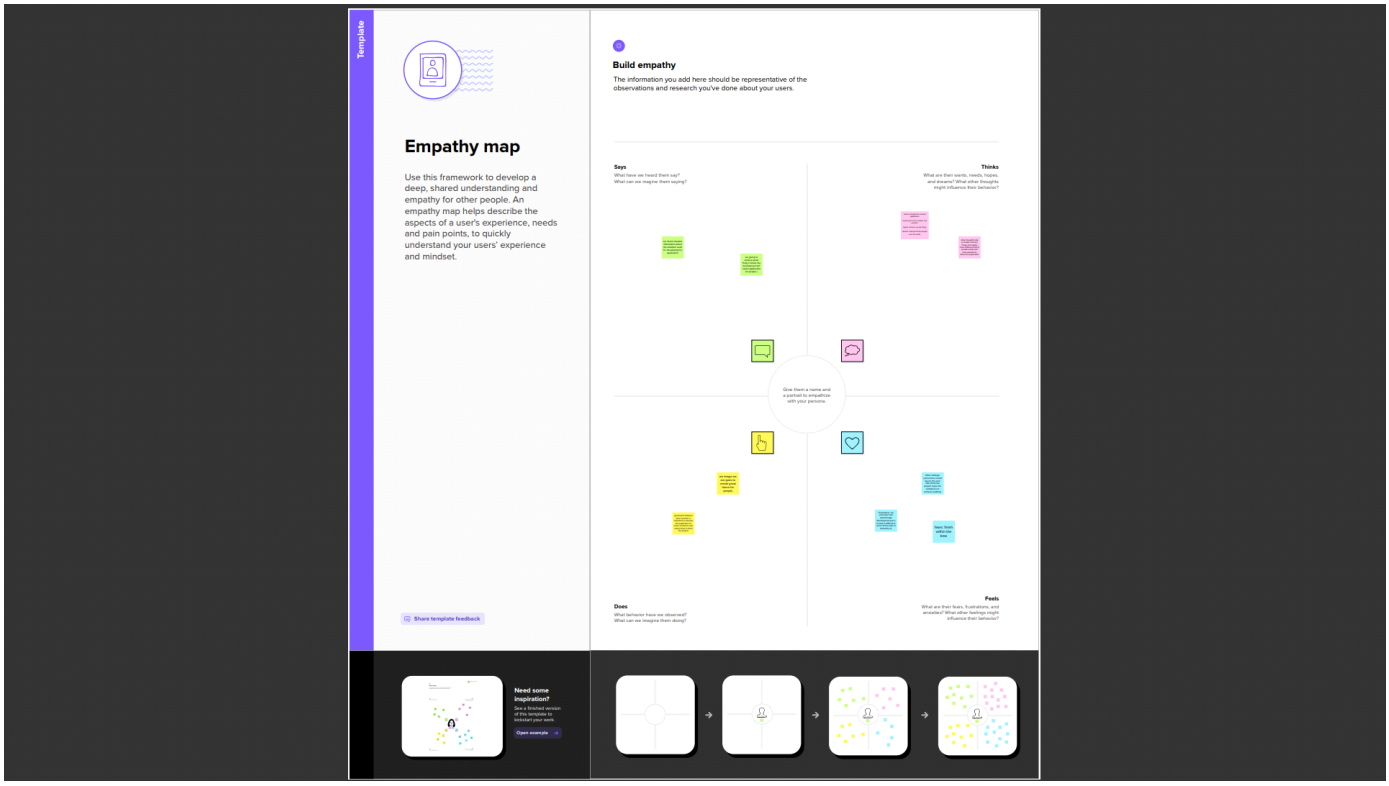
Having a personal finance app allows the users to feel more confident and secure with their finances as they are exactly in the know-how of it. It is useful in distributing money between must-buys and occasional luxuries. And at the same time, a certain sum of money can be kept aside for saving.

Helps Make Sound Financial Decisions

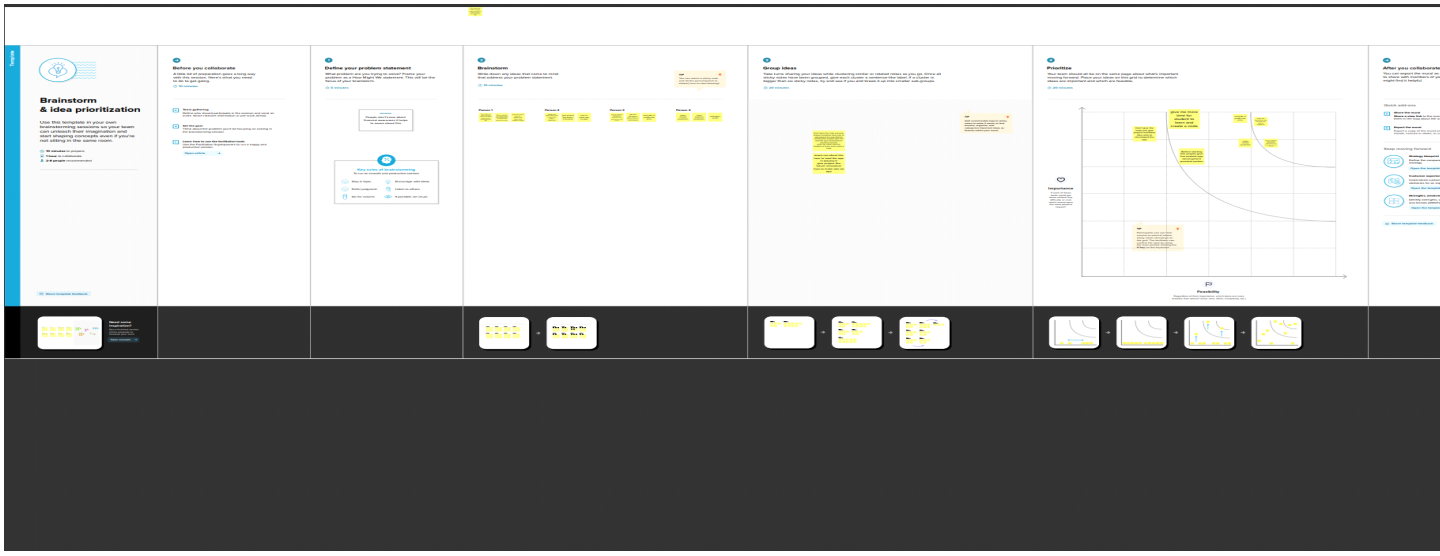
A further benefit of good money management apps is that they prevent impulsive purchases. The app allows for proper monthly budget planning and funds distributions, and with this, the risk of wasting money is reduced tremendously. This way, one can save for things they really need instead of spending it in the wrong places.

Problem definition and Design thinking:

Empathy map:

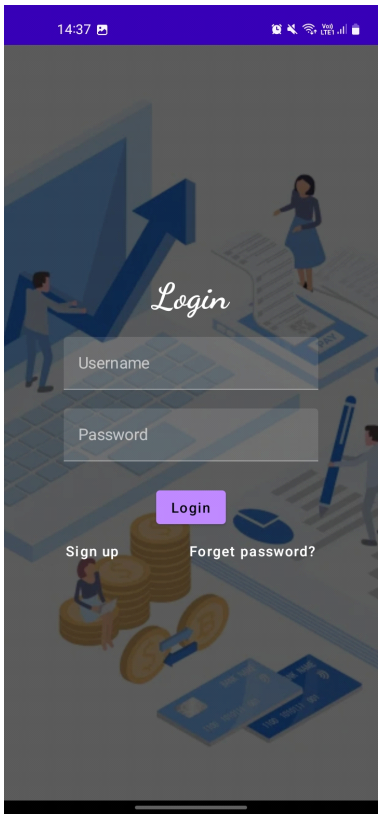


Ideation & Brainstorming map:

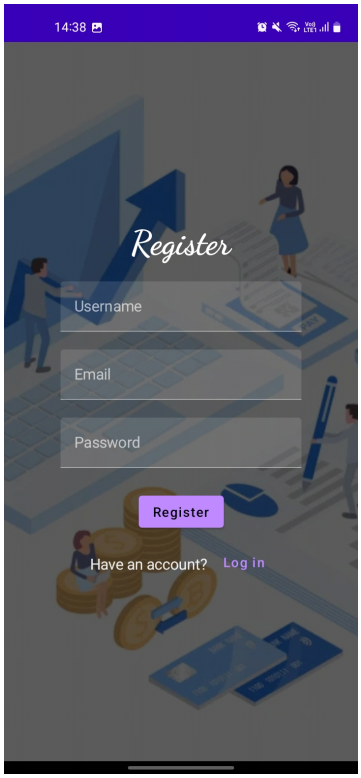


Result:

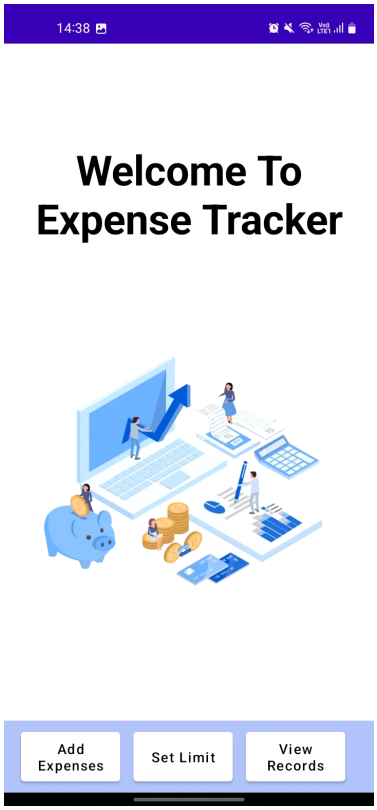
Login page:



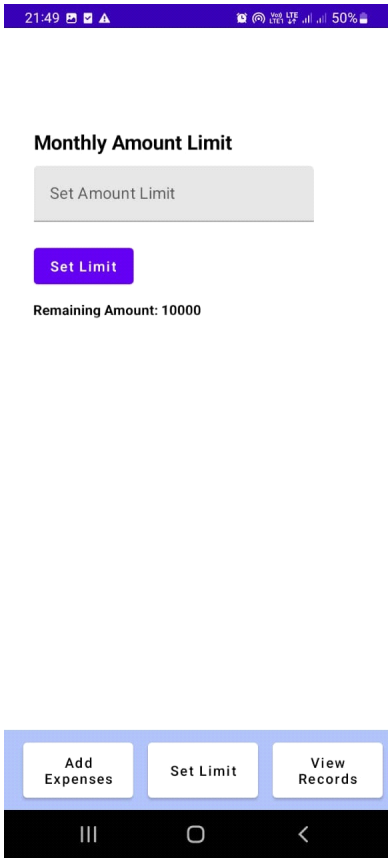
Register page:


14:38
Register
Username
Email
Password
Register
Have an account?
Log in

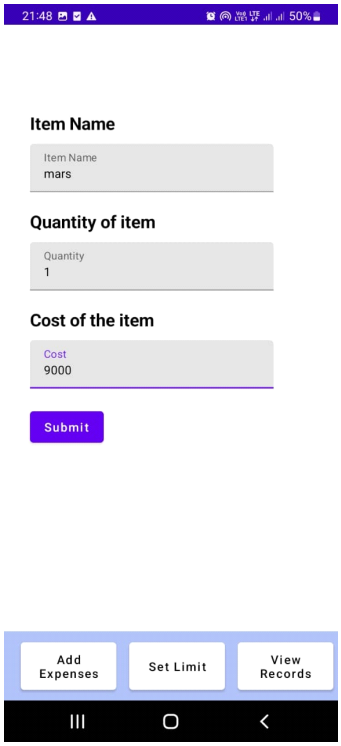
Main page:



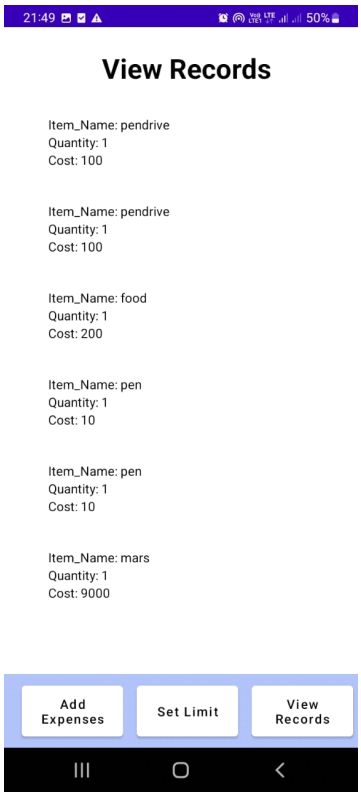
Set Limit Page before adding any data in expenses:



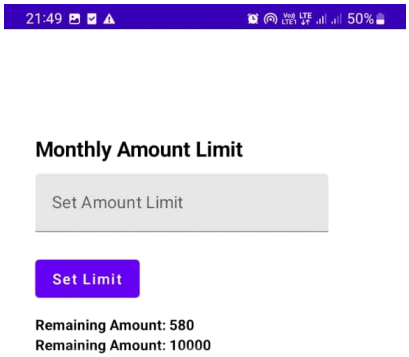
Purchase entry page:

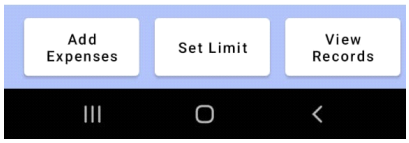


Record of purchase:



Set limit after purchase:





Advantage:

Sound financial management leads to increased visibility within the operations, and it supports understanding of the numbers at each level in the business or institute. The advantages of financial management make sure there is investor confidence. Investors are usually keen to look for signs of security within business operations. Effective financial management allows for the correct balance between risk and profit maximization.

Financial management also endorses better decision making. When the relevant facts are easily accessible because of digitization and organization, it becomes easier to derive solutions based on the circumstances of the situation.

As an incredible benefit, financial management assists with taxation. Taxes have often been frowned upon as one of the limitations of financial system. There are tax loopholes and exceptions for enterprises and institutes which can be taken advantage of if the terms are satisfied. For example, a business can claim tax deductions based on their quantity of office space.

Disadvantage:

The more you use your mobile phone daily, the more you eat into your available data allowance and memory space. If you have a super-duper smartphone with additional SD cards and an awesome data plan, this may not be a problem, but it is still something you need to keep in mind when making the choice of whether to use finance apps or not.

As the technology and protective measures that are put in place to keep your financial details and personal information falling into the wrong hands keep improving, cybercriminals are learning how to beat the systems in place and finding faults in new software and apps.

Application:

A financial management app can be used in many ways to help users manage their finances more effectively. Here are a few examples:

- Budgeting: A financial management app can help users create and stick to a budget by tracking their income and expenses. Users can set spending limits for various categories, such as food, transportation, entertainment, and more. The app can provide notifications when the user is approaching their spending limit, helping them avoid overspending.
- Expense tracking: By logging expenses in the app, users can get a clear picture of where their money is going. This information can help them identify areas where they can cut back on spending or adjust their budget.
- Investment management: financial management apps offer investment management tools, such as portfolio tracking and asset allocation advice. Users can monitor their investments and adjust based on market trends and their personal financial goals.
- Debt management: A financial management app can help users track their debt, including credit card balances, loans, and mortgages. Users can set goals for paying down their debt and receive notifications when they make progress towards those goals.

- Savings goals: Users can set savings goals in the app, such as a down payment for a house, a vacation fund, or an emergency fund. The app can help them track their progress towards those goals and offer suggestions for ways to save more.

Overall, a financial management app can be a powerful tool for helping users take control of their finances and achieve their financial goals.

Conclusion:

This project provides more creative and innovative ideas for us. To know lots of things about android and Finanancial management techniques. It's helpful for creating bond between team members.

Future Scope:

In future we will try to represent the financial expense using graphical representation using charts.

Using ML to predict the price of product on particular season.

Give tips to user for investment

To create self-awareness among people about financial management.

Give tips for entrepreneur about the product sales and prove people interest rate about the product.

Appendix:

Adding dependency in build.gradle(Module: app) file in project

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

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

    defaultConfig {
        applicationId "com.example.expensestracker"
        minSdk 22
        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'
```

```
    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"

    // Adding Room dependencies
    implementation 'androidx.room:room-common:2.5.0'
    implementation 'androidx.room:room-ktx:2.5.0'

}
```

From the folder com.example.expensestracker file create AddExpensesActivity.kt

```
package com.example.expensestracker

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")
        }

    }
}
```

Adding Expense class file:

```
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?,
)
```

Adding ExpenseDao interface file:

```
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)
}
```

Adding ExpenseDatabase class file:

```
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
            }
        }
    }
}
```

Adding ExpenseDatabaseHelper.kt class file:

```
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
    }




}
```

Adding Items class file

```
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?,
)
```

Adding ItemDao interface file:

```
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)
}
```

Adding Expensetracker abstruct class file:

```
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
        }
    }
}
}
```