A project Report on

MONEY MATTERS : A PERSONAL FINANCE MANAGEMENT APP

BACHELOR OF COMPUTER SCIENCE

Devanga Arts College,

(Affiliated to Madurai Kamaraj University)

Aruppukottai-626101.

Submitted

By

K.SURESH (Reg No:C0S42021)

S.RAJESH KANNA (Reg No:C0S42017)

B.SABARINATHAN (Reg No:C0S42020)

Under the Guidence of

Mrs VALARMATHI M.Sc.,

Devanga Arts College

(Affiliated to Madurai Kamaraj University)

Aruppukottai-626101

1.INTRODUCTION

This module was designed to introduce you to a variety of money matterst ideas and concepts to build your knowledge and give you confidence in taking control of your finances.

You will begin by learning about our 4 core money management pillars to get a better understanding of your own money personality and then you will start to identify your spending habits, work on creating realistic goals, plan for debt repayment and practice budget management. There are six sections, each spanning ten to fifteen minutes, and every section is packed with interactive learning material that concludes with a knowledge check quiz to confirm your understanding of the key concepts.

1.1.OVERVIEW

Money matters apps are software applications designed to help users manage their personal finances. These apps typically include features such as budget tracking, expense tracking, bill payment reminders, and account matters.

Some common features of money matters apps include:

Budgeting: Many money matters apps allow you to create custom budgets based on your income and expenses. These apps help you track your spending against your budget and provide alerts when you exceed your budget for a particular category.

Expense tracking: Money matters apps help you keep track of your expenses by automatically categorizing your transactions and providing a comprehensive overview of your spending habits. You can also manually add expenses to the app.

Bill payment: Some money management apps allow you to pay bills directly through the app, with features like scheduled payments, payment reminders, and the ability to link your bank accounts.

Account management: You can link your bank accounts, credit cards, and other financial accounts to the app, giving you a consolidated view of your financial accounts. You can also view account balances, transaction history, and transfer funds between accounts.

Investment tracking: Some money matters apps allow you to track your investments, including stocks, mutual funds, and retirement accounts.

Financial education: Many money management apps offer financial education resources and tools, such as articles, videos, and calculators, to help you better understand your finances and make informed decisions.

Money matters is the process of organizing and managing your finances, so you can achieve your financial goals and improve your overall financial well-being. It involves making informed decisions about your income, expenses, savings, investments, and debt. Here are some specific examples of how you can use money matters:

1.2 PURPOSE

Budgeting: Creating and sticking to a budget is an essential part of money matters. This involves tracking your income and expenses, setting spending limits, and making adjustments as needed.

Saving: Saving money is an important part of money management. This involves setting aside money for emergencies, retirement, and other long-term goals.

Investing: Investing your money can help you grow your wealth over time. This involves researching investment options and making informed decisions based on your financial goals and risk tolerance.

Debt matters: Managing your debt is an important aspect of money management. This involves paying off high-interest debt, avoiding new debt, and managing your credit score.

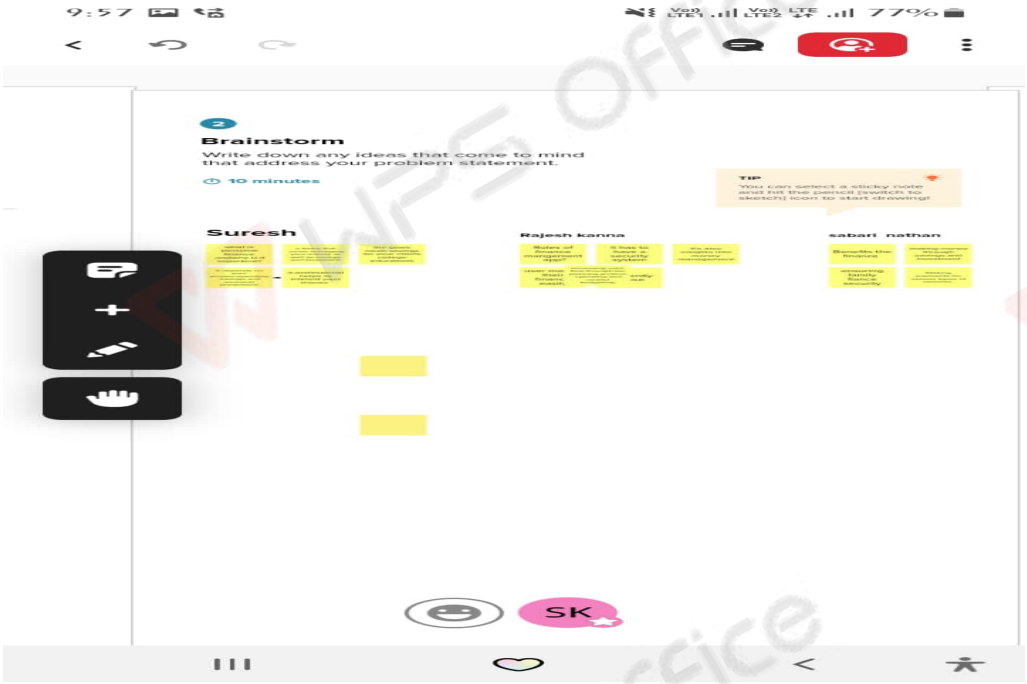
Retirement planning: Planning for retirement is an important part of money management. This involves saving for retirement, understanding your retirement benefits, and making informed decisions about your retirement savings.

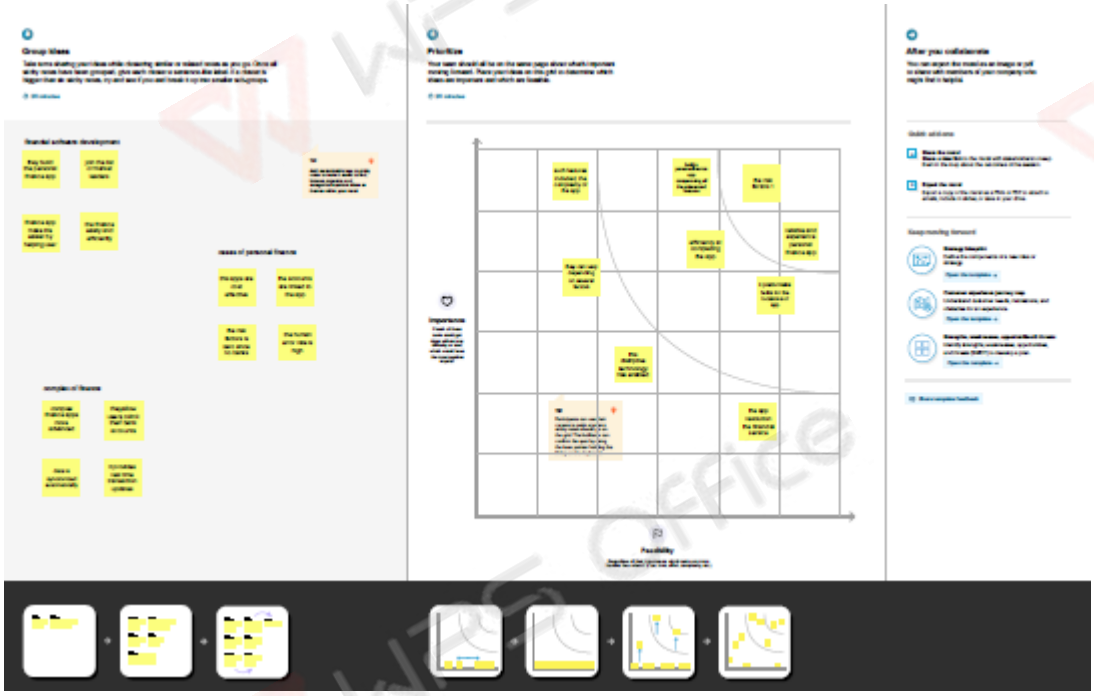
Financial education: Improving your financial literacy is an important part of money matters. This involves learning about personal finance topics, such as budgeting, investing, and debt management.

2.PROBLEM DEFINITION & DESIGN THINKING

2.1 EMPATHY MAP

2.2IDEATION& BRAINSTROMING MAP





3.ADVANTAGES & DISADVANTAGES

Money matters apps have several advantages and disadvantages.

Advantages:

1. Convenient: Money matters apps allow users to manage their finances conveniently from their smartphones or tablets. Users can track their expenses, set budgets, and monitor their investments on-the-go.
2. Automated Tracking: These apps automatically categorize transactions, which helps users understand where their money is going.
3. Budgeting: Money matters apps allow users to set budgets and receive alerts when they exceed their budget. This feature helps users to stay on top of their finances and avoid overspending.
4. Saving Goals: Many money matters apps have a feature that allows users to set saving goals. Users can set a target amount, and the app will help them track their progress towards achieving their goal.
5. Financial Planning: Some money matters apps provide financial planning tools that can help users plan for their future, including retirement planning and investment advice.

Disadvantages:

1. Security: Money matters apps require users to link their bank accounts, which can be a security risk. If the app is hacked, users' financial information could be compromised.
2. Fees: Some money management apps charge fees for premium features or for linking to certain financial institutions.
3. Inaccuracies: Money management apps may not always categorize transactions correctly, which can lead to inaccurate budgeting and financial planning.
4. Reliance on Technology: Money management apps require users to have access to technology, which can be a barrier for some people, especially those who are not familiar with using smartphones or tablets.
5. Limited Support: Some money management apps may not have adequate customer support, which can be frustrating for users who encounter issues or have questions.

Money is a medium of exchange that is widely accepted in transactions for goods and services. It is a basic economic concept that is used to facilitate trade and commerce. There are different forms of money, including physical cash, coins, bank deposits, and digital currency. Physical cash and coins are tangible forms of money that can be used to make purchases in-person. Bank deposits, on the other hand, are digital representations of money that can be accessed through a bank account, and can be used to make purchases online, through a debit or credit card, or through electronic funds transfer.

Digital currency, such as Bitcoin, is a type of decentralized and digital currency that operates on a blockchain network. It is not backed by a government, and its value is determined by market forces.

Money serves several functions, including as a medium of exchange, a unit of account, a store of value, and a standard of deferred payment. As a medium of exchange, money allows people to buy goods and services from each other. As a unit of account, money

provides a common standard for measuring the value of goods and services. As a store of value, money can be saved and used later to make purchases. As a standard of deferred payment, money allows people to make purchases on credit, with the understanding that they will pay for the purchase at a later time.

The study of money and its use in the economy is called monetary economics. Monetary policy, which is set by central banks, is used to control the supply of money in an economy and influence economic activity.

4.APPLICATION

A money matters application is a software tool that helps users manage their personal finances. These apps allow users to track their income, expenses, savings, and investments in one place, and provide insights into their spending habits and financial health. Money matters apps are available for desktop computers, smartphones, and tablets, and are often free or low-cost.

Features of money matters apps can include:

1. Expense tracking: These apps allow users to track their expenses, categorize them, and see where their money is going.
2. Budgeting: Money matters apps allow users to set budgets for different categories, such as groceries, rent, or entertainment, and track their spending against those budgets.
3. Savings goals: Many money management apps have a feature that allows users to set savings goals, such as saving for a down payment on a house or a vacation, and track their progress towards those goals.
4. Investment tracking: Some money management apps allow users to track their investments, including stocks, bonds, and mutual funds, and provide insights into their portfolio's performance.
5. Bill payment reminders: Money management apps can alert users when bills are due, helping them avoid late fees and missed payments.
6. Financial advice: Some money management apps provide financial advice and insights based on a user's financial data, such as recommending ways to save money or invest more wisely.

Some examples of popular money management apps include Mint, Personal Capital, YNAB (You Need A Budget), PocketGuard, and Clarity Money.

5.CONCLUSION

In conclusion, money matters is an essential skill for individuals to maintain financial stability and achieve their financial goals. Effective money management involves creating a budget, tracking expenses, saving for emergencies and future goals, and investing wisely. Money matters applications can be helpful tools to streamline these processes, automate financial tracking, and provide insights into spending habits and investment performance. However, it is important to be aware of the potential risks and limitations of these apps, such as security concerns and inaccuracies in categorizing transactions. Overall, developing good money management habits and utilizing appropriate tools can help individuals take control of their finances and achieve financial well-being.

6.FUTURE SCOPE

The future scope of money matters is promising, as advances in technology continue to revolutionize the way we manage our finances. Here are some potential areas of growth:

1. Artificial intelligence: Money management apps may utilize artificial intelligence (AI) to provide personalized financial advice and insights based on a user's financial data. AI can also help identify patterns and trends in spending habits and investment performance, allowing for more accurate predictions and recommendations.
2. Blockchain technology: Blockchain technology has the potential to transform the way we store and transfer money, as well as make financial transactions more secure and transparent.
3. Financial wellness: There is a growing focus on financial wellness, which goes beyond traditional money management to include overall financial health and well-being. Money management apps may incorporate features such as financial coaching, debt management, and retirement planning to help users achieve their financial goals.
4. Integration with other services: Money matters apps may become more integrated with other services, such as banking, investing, and insurance, to provide a more comprehensive financial management experience.
5. ESG investing: Environmental, social, and governance (ESG) investing is becoming increasingly popular, and money management apps may incorporate ESG investment options to cater to users' ethical and sustainability preferences.

Overall, the future of money management is exciting, with technology playing a significant role in transforming the way we manage our finances and achieve financial well-being.

7.APPENDIX

```
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?,
                                    )
```

```
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)
                               }
```

```
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
                                    }
                                }
                            }
                        }
```

```
package
com.example.expensest
racker
                        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
            }
        }
```

```
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?,
                )
```

```
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)
}
```

```
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
            }
```

```
                }
            }
        }
```

```
package
com.example.expensestra
cker
```

```
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_QUAN
TITY)),
                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_QUAN
TITY)),
```

```
                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_QUAN
TITY)),
                    cost =
cursor.getString(cursor.getColumnIndex(COLUMN_COST
)),
                )
                item.add(items)
            } while (cursor.moveToNext())
        }
        cursor.close()
```

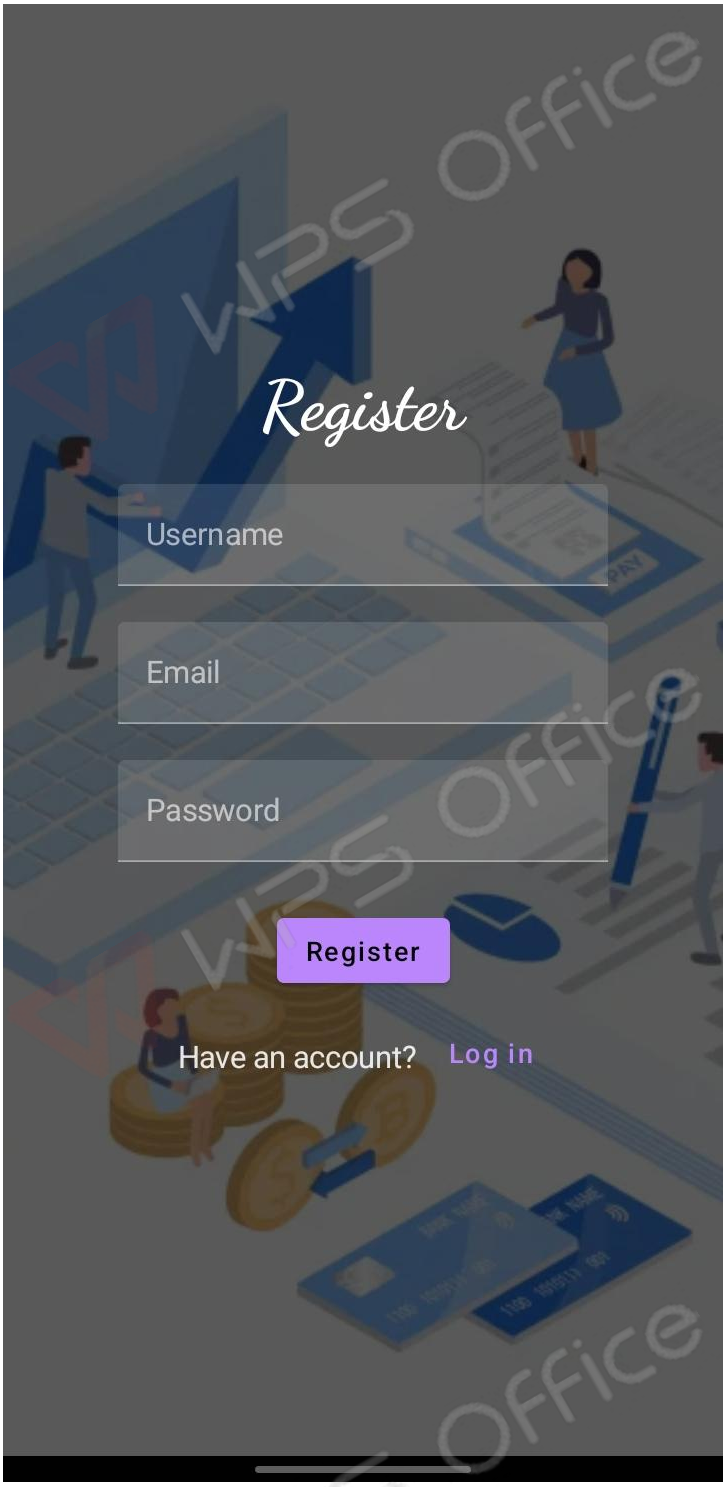
```
            db.close()
            return item
        }
    }
```

```
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?,
)
```

Login Page :

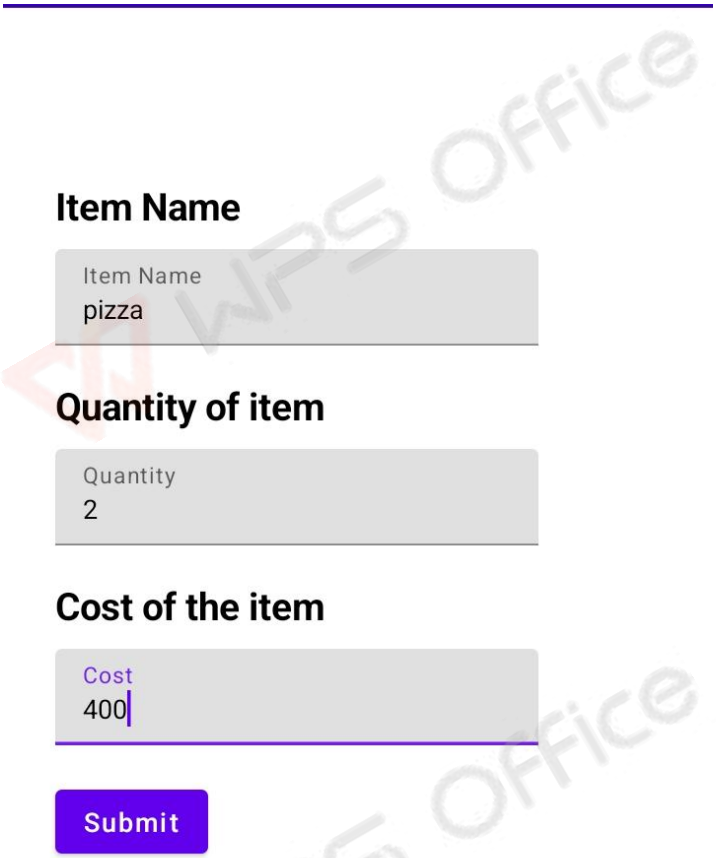
RegisterPage :


Register
Username
Email
Password
Register
Have an account?
Log in

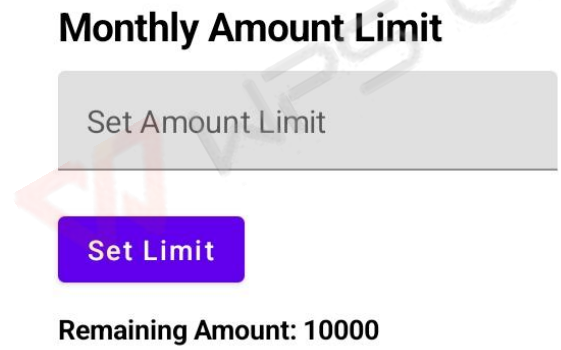
MainPage :

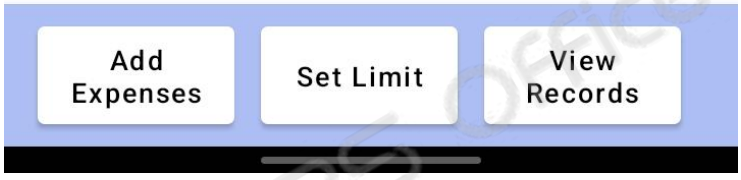
Welcome To Expense Tracker

Add Expenses page:

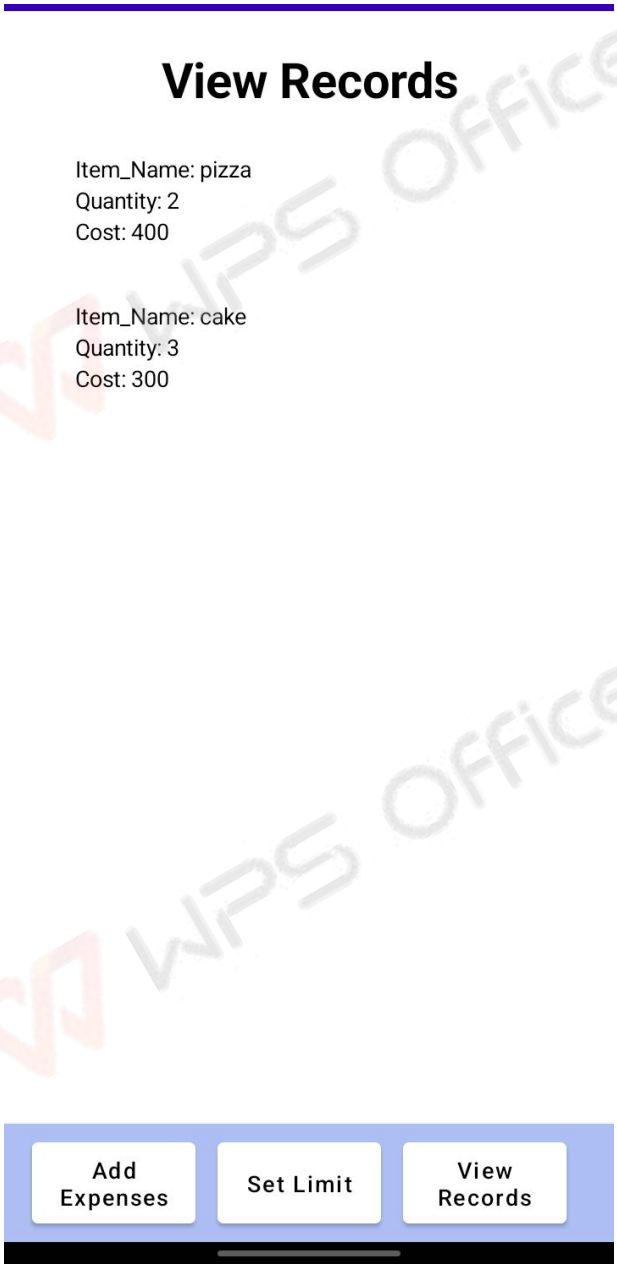


Set Limit Page before adding any data in expenses:





View Records Page:



Set Limit Page After adding expenes in add expense page:


Monthly Amount Limit
Set Amount Limit
Set Limit
Remaining Amount: 9300
Add Expenses
Set Limit
View Records