ARULMIGU PALANI ANDAVAR COLLEGE OF ARTS AND CULTURE

PROJECT DOCUMENTATION FOR MUDHALVAN SCHEME

MONEY MATTERS: PERSONAL FINANCE MANAGEMENT APP

1.INTRODUCTION:

1.1:Overview:

Finance apps make life easier by helping users manage their finances easily and efficiently. The best personal finance app not only helps users with monitoring, budgeting, accounting and expense tracking but also gives helpful insights into money management. It also gives users various investment options, tax advice, insurance inputs, and above all, a proper security system.

The extensive role of a personal finance app makes them secure a robust place in the list of Top Fintech Trends.

So, if you are also planning to build a FinTech app for your business, our latest blog on How Much Does It Cost To Develop A Fintech App will help you plan the development budget effectively.

1:2:Purpose:

Put simply, it is a piece of software that allows people to control their finances through an online interface or via a mobile device like a smartphone or a tablet.

People use money management apps for a number of purposes:

- **Monitoring spendings and earnings,**
- **Saving money and staying on budget,**
- **Regular balance checks,**
- **Online transactions and payments,**
- **Smart deposits and investments, and**
- **Insights and advice about personal finances.**

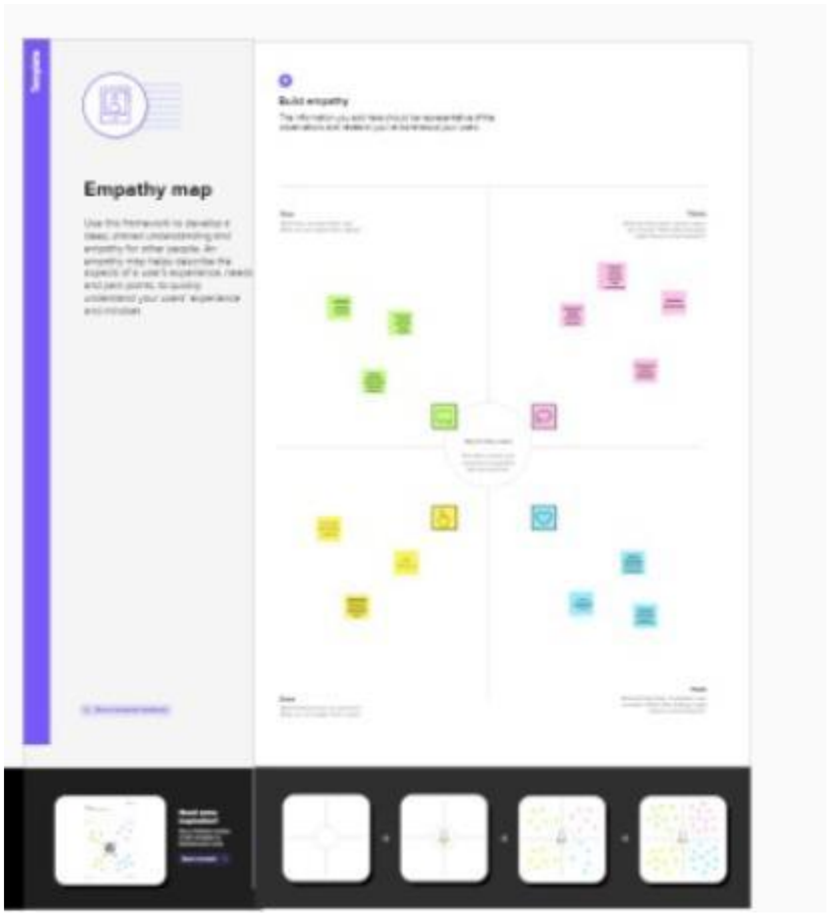
Personal finance apps like Mint have been around for less than a decade, however, they boast an impressive user base and are attracting new users willing to enjoy the obvious benefits:

- **A comprehensive view of all their financial transactions,**
- **The ability to set financial goals and see them through to completion,**
- **Easy budgeting and savings,**
- **An awareness of one's financial behavior patterns, and**
- **Learning the basic principles of personal money management.**
- **Now that we've covered the benefits of these solutions, let's look closely at the existing types of consumer-oriented money management software.**
  Product in the financial market:

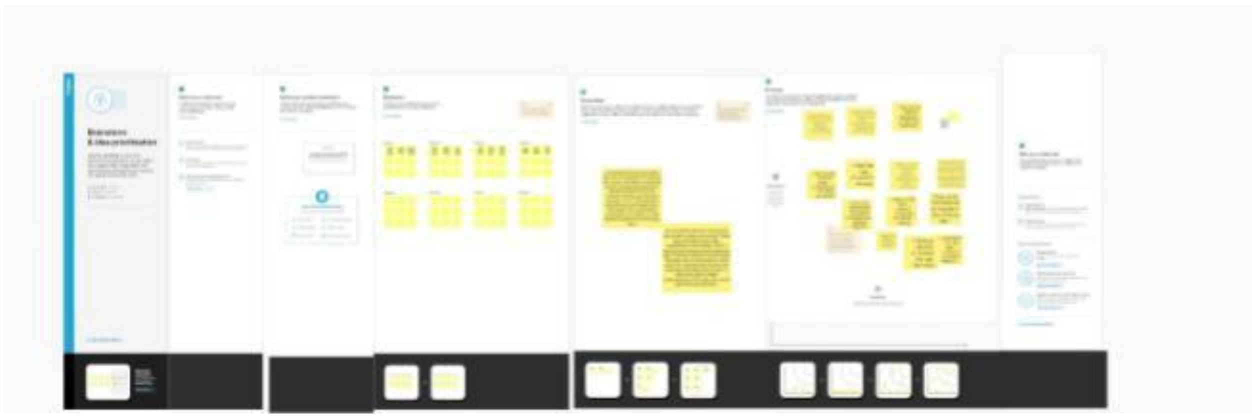
  Mint. The budget planner app allows connecting bank accounts and credit cards to keep track of your income and expenses. It provides spending-based budget targets, including an overview of the daily budget. These goals may be changed and expanded in the future.
  Mvelopes. It is another alternative to the Mint that uses digital envelopes to help users control and manage their capital. The app helps handle your finances and save a part of your income (approximately 10%) from otherwise being spent.
  EveryDollar. The app lets users visualize their income and expenses to quickly analyze and manage their finances. They can also add budget-appropriate categories and see where they go over or under budget.
  PocketGuard. Using PocketGuard, users can link all their bank accounts in one place and keep track of their income and expenses. This Mint alternative will highlight how much money they have for spending and notify them if they go over budget.
  You Need a Budget (YNAB). It is an individual finance and spending tracker application that can save up to $600 in the first two months and over $6000 in the first year.

2:PROBLEM DEFINITION AND DESIGN THINKING:

2.1 EMPATHY MAP:

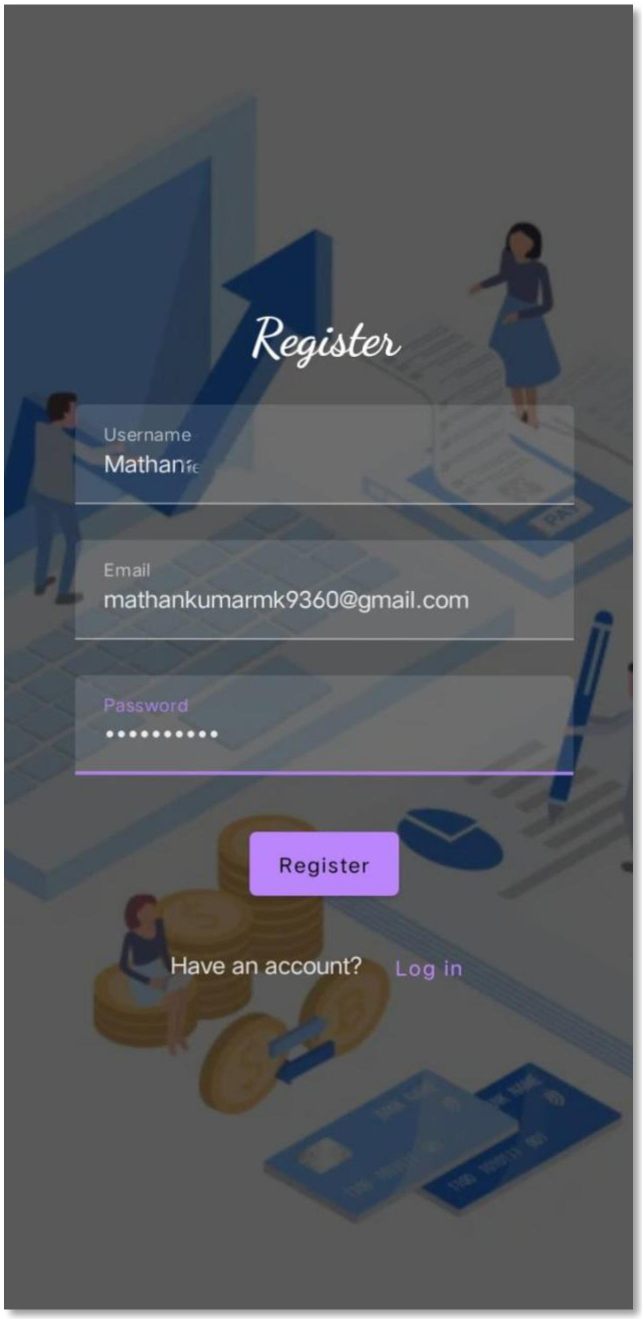


2.2IDEATION & BRAINSTORMING MAP:

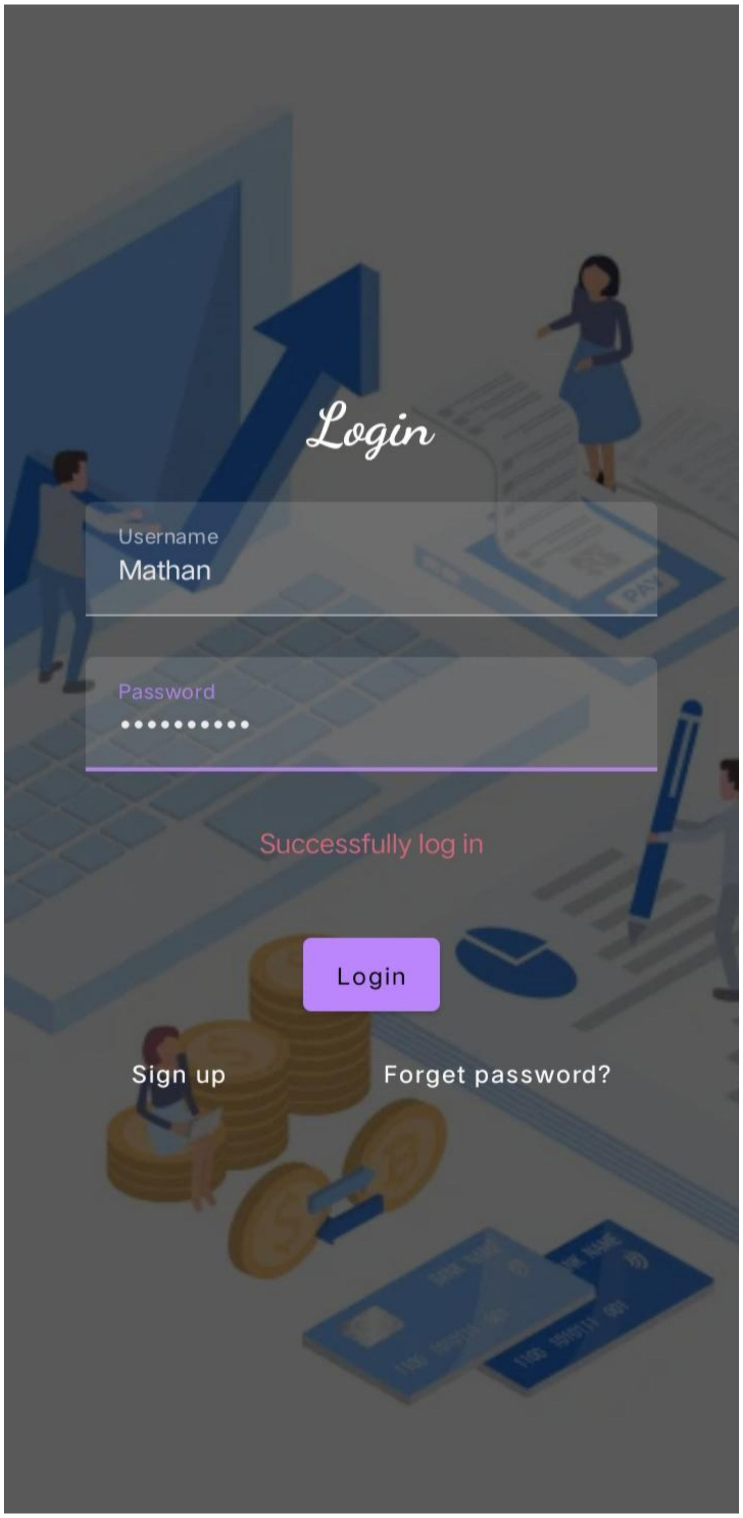


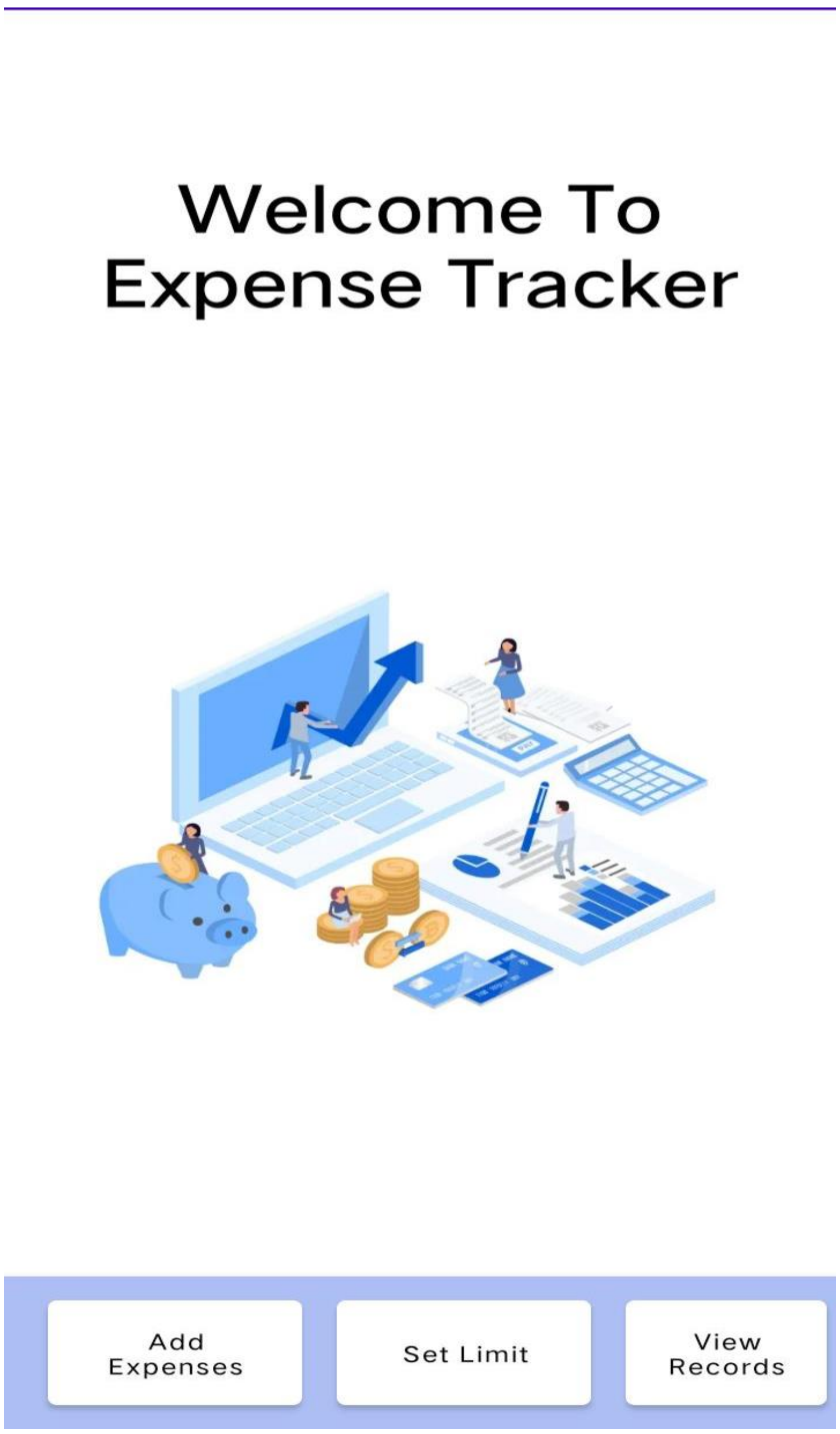
3:RESULT:

Register Page:



Log-in Page:

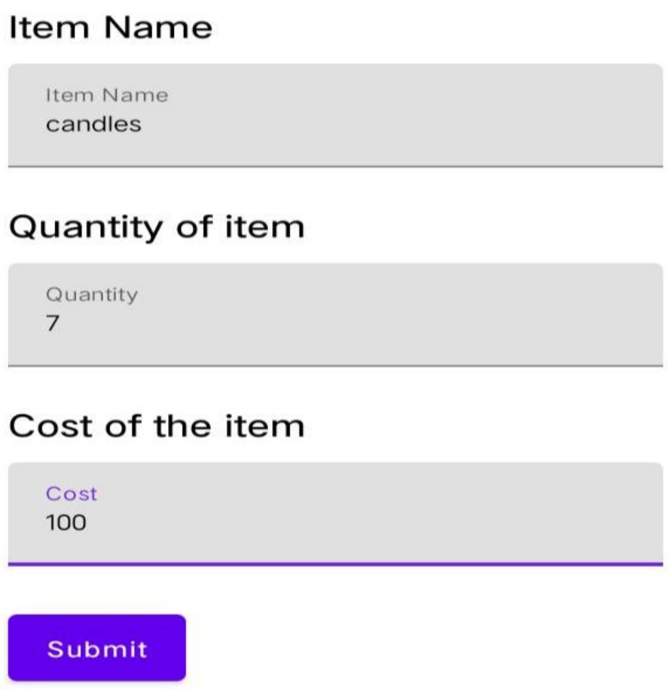

Login
Username
Mathan
Password
••••••••••
Successfully log in
Login
Sign up
Forget password?



Welcome To
Expense Tracker

Home page:

Expanses:


Item Name

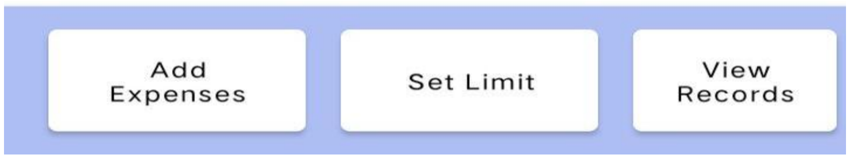
Item Name
candles

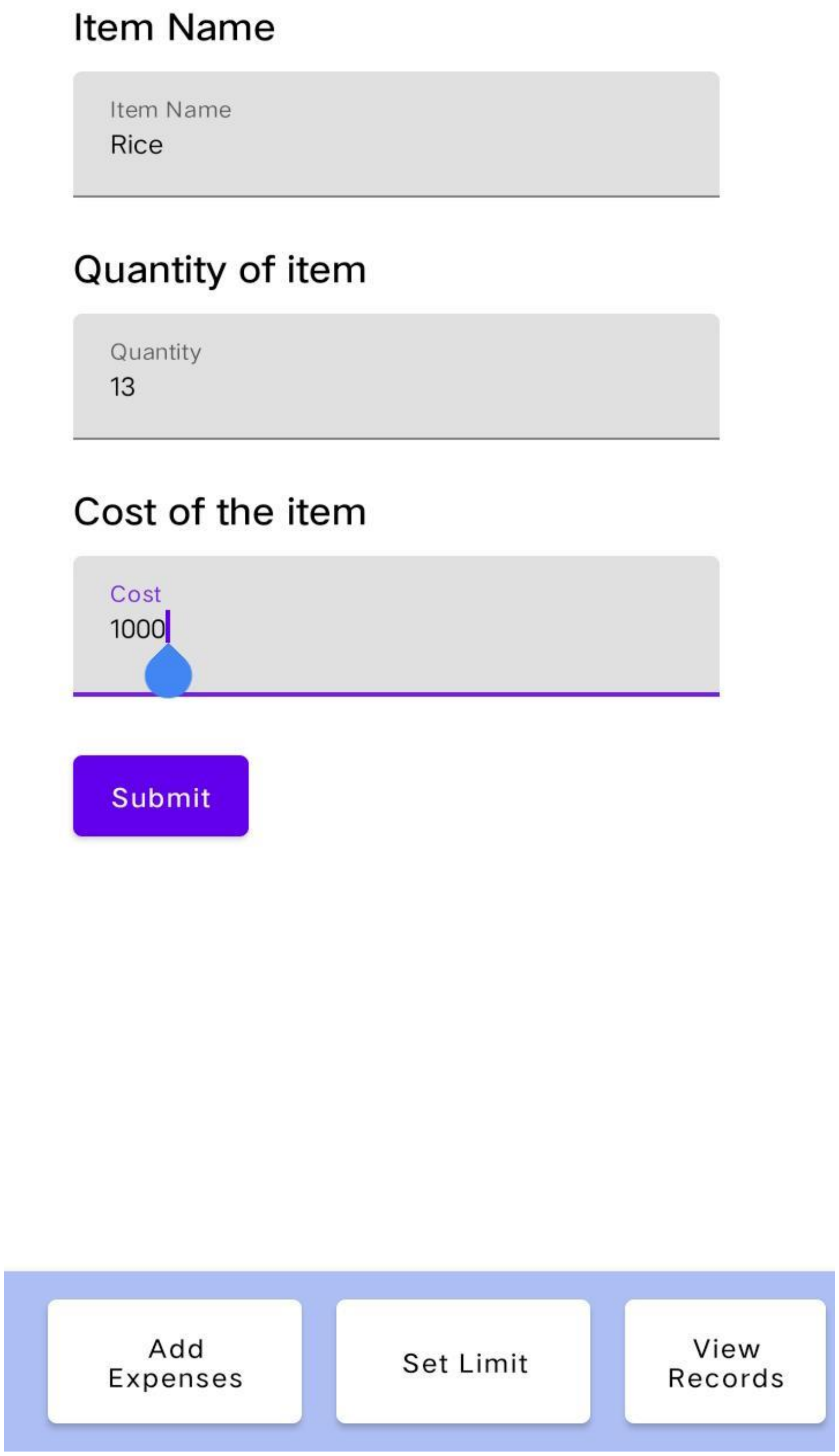
Quantity of item

Quantity
7

Cost of the item

Cost
100





Item Name

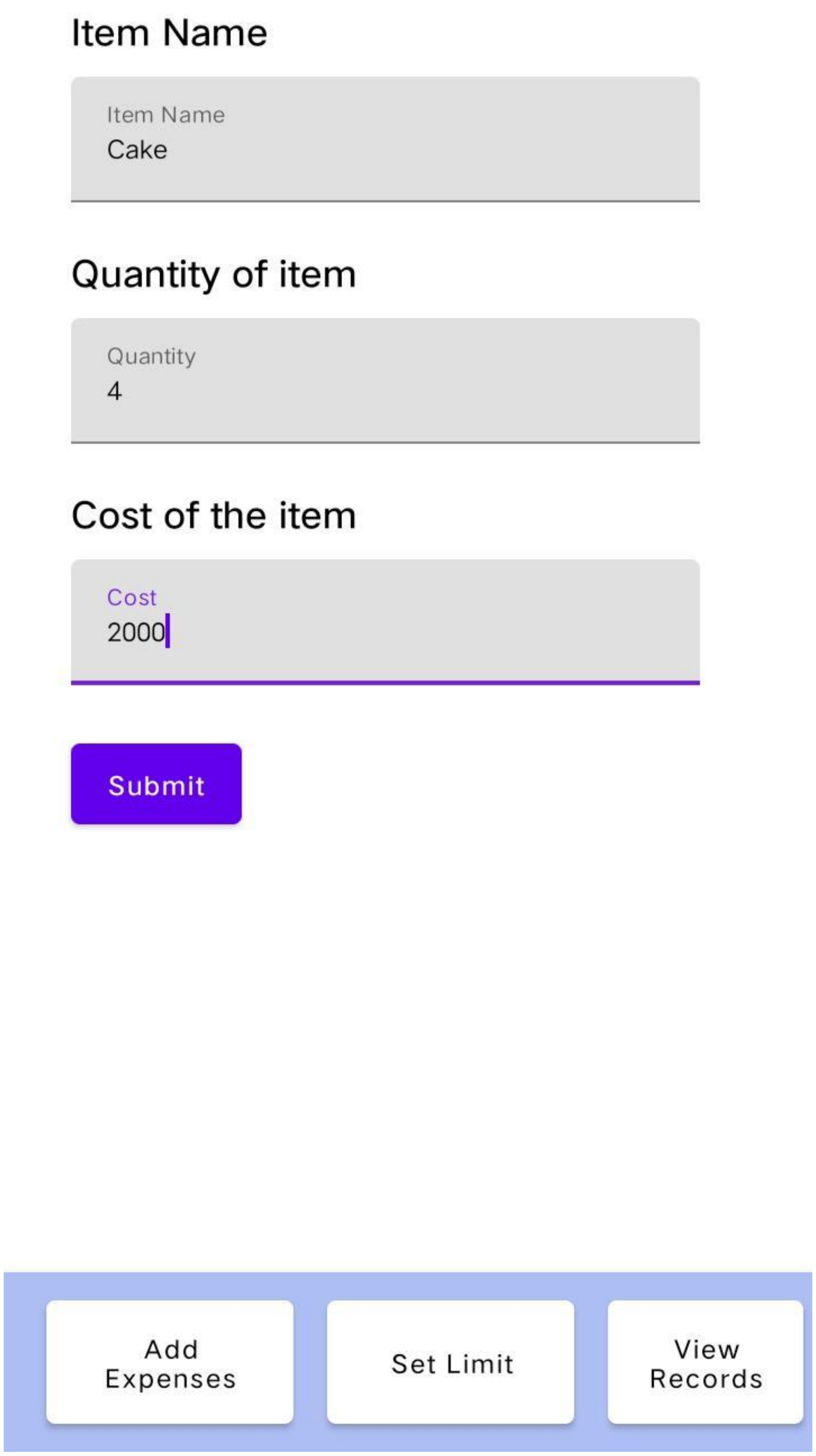
Item Name
Rice

Quantity of item

Quantity
13

Cost of the item

Cost
1000


Item Name

Item Name
Cake

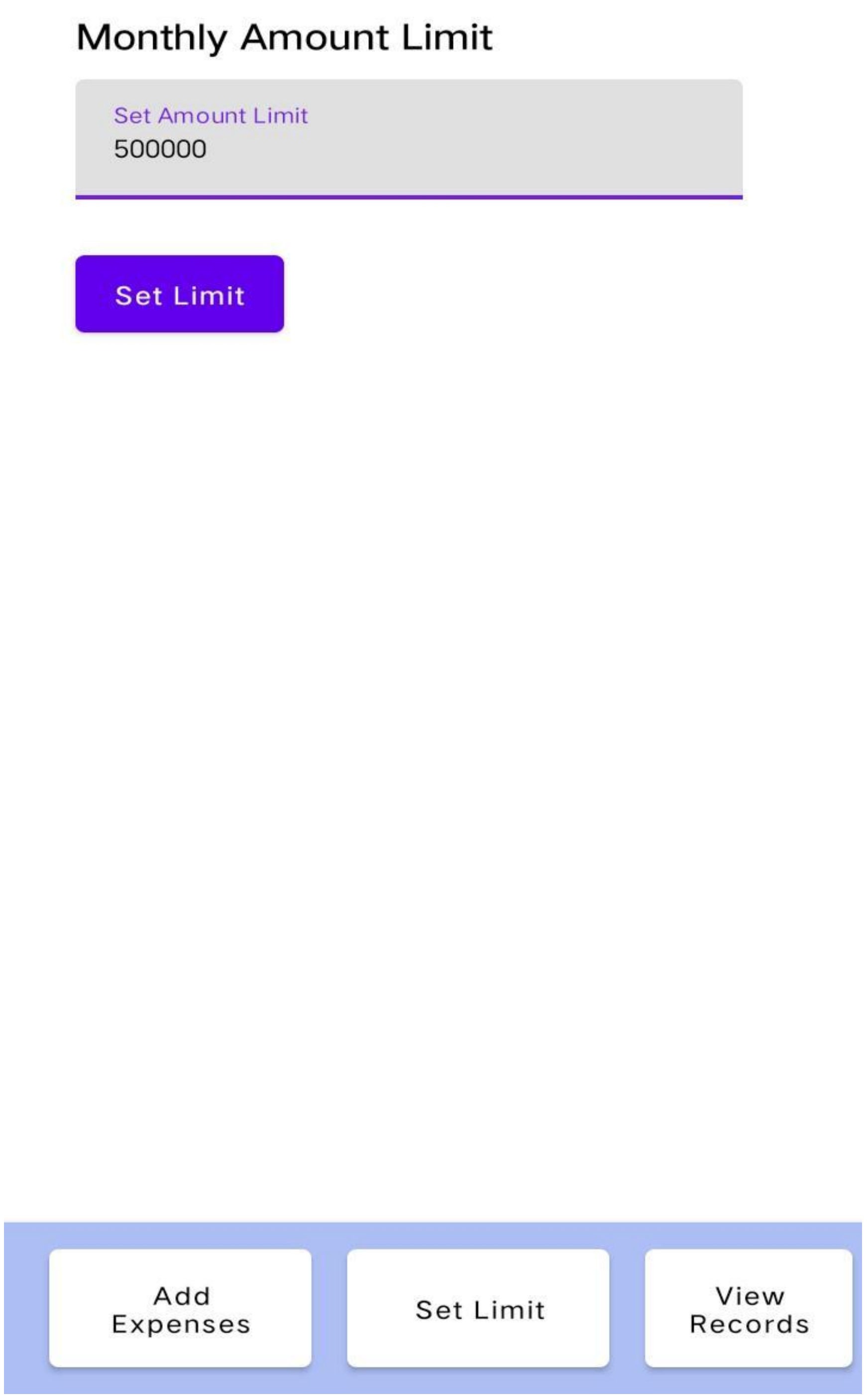
Quantity of item

Quantity
4

Cost of the item

Cost
2000

Set limit:



Monthly Amount Limit

Set Amount Limit
500000

4:ADVANTAGES:

1. Access info from anywhere

Probably the biggest benefit of financial apps is their accessibility. Whether at work, traveling, or far from a computer, users can make changes and updates right from their mobile device, as long as they have an internet connection.

2. Deposit checks

Many bank apps now allow users to deposit a physical check through the app. You simply take a photo of the front and back of the check, upload it, and the deposit is submitted. This saves a lot of time instead of having to do it in person.

3. Save time

It can be quite a pain to have to travel to a bank, which is usually only open during regular working hours, just to make a transaction or view an account balance. With a financial app, everything can be done whenever a user thinks about it, and they don't have to go anywhere. You don't have to wait in long lines at the bank just to make a deposit or an online payment.

4. Stay organized

When people can keep tabs on their finances daily with information right at their fingertips, they're more likely to stay on top of personal finances and stay organized. Apps allow for nearly real-time accuracy so there's never any question about an account balance. All income and expenses can be monitored with ease and efficiency.

5. Manage business finances

Apps aren't just for personal finances. Financial apps help business owners manage accounting, reporting, and cash flow. Budgets can be set up and tracked, payable and receivable accounts can be monitored, and reports can be pulled and viewed in moments. These benefits help businesses stay on track, and the added visibility helps them make better business decisions.

6. Access everything in one place

Aside from banking and financial institution apps where you can make transactions, there are also budgeting and financial management apps that combine all your open accounts in one place, including retirement savings and other investments, checking and savings accounts, credit cards, assets, and debts. It's much easier to view your financial standing with all this information in one simple app.

7. Use automation

Many of these apps also send push notifications to mobile devices. This means users can be alerted if there is suspicious activity on an account, an account balance is getting dangerously low, or their credit score is going up or down. Users can also set up alerts and reminders for important dates like bill due dates and tax deadlines so they never miss anything.

8. Monitor goal progress

By setting up goals on financial apps, users can stay on top of their progress. This boosts motivation, and it's very satisfying to see a savings account balance increase or a debt account decrease each month. Apps allow users to create and track goals and make changes as needed.

This kind of app is much more all-inclusive than a simple financial app. It offers many features to its users and allows them to perform many functions instead of just a few basic ones.

This app is very efficient because it saves users time.

Real-time updates are one of this app's most significant benefits, so your user is constantly updated about everything happening.

9.Analytics and reports

Using this feature, users can keep track of their financial activities to manage and save their money properly. Due to well-organized data, users can get daily, weekly, monthly, and yearly reports to see income, expenses by category, summaries of payees, forecasts, etc.

10.Synchronization

Users of a budgeting app will want all their money to be easily accessible on the same digital platform to view information and data about their financial situation. Your application must sync all user accounts, debit and credit cards, etc., to get the relevant information. This allows consumers to view info and data from all accounts in one digital environment and better manage their budget.

11.Registration and account creation

Any app starts with registering and creating an account. But when it comes to finances, authorization and security require special attention. The best practices for protecting sensitive user data are two-factor authentication, fingerprint or voice recognition, and the generation of a unique security code.

12.Budgeting and expense categorization

You should provide a budgeting option for users to plan their spending for a week, a month, or several months. Let your users set spending limits, and the app will notify them before overspending. Another awesome feature is to show your users exactly what they spend their money on. A budgeting app can provide categories for expenses and automatically categorize users' transactions to track their spending effectively.

l funds, IRA investments, etc.

13.Investment tracking

The feature lets users track where costs go and where to invest their money. An investment tracker can help users stay on top of market benchmarks and monitor all their asset allocations, such as brokerage accounts, real estate investments, mutual funds, IRA investments, etc.

14.Notifications and alerts

This feature is required for the financial application, so we recommend adding it to the functionality. But make sure the reminders are timely and non-intrusive. You can also consider other push notification scenarios, for example, notifying your users when their goals are met or reminding them to stay on budget

There is almost zero risk factor here because none of your bank accounts need to be linked to the app.

Because of a straightforward app development process and simple features, this app is inexpensive.

DISADVANTAGES:

The main disadvantages of financial management is that it does not provide accurate information. This implies that you may need to examine your financial decisions on a regular basis to ensure that you have kept them up to date in the event that anything has changed.

Creating a strategy to describe the aims and objectives of any firm, as well as determining the sources of money that will be used to fulfil the company's daily operating expenditures, is the definition of financial management. However, there are some significant disadvantages of financial management as well.

- **Changes in government policy might occur suddenly at any time, and these changes can have an impact on the financial strategy.**
- **Because of a lack of coordination and communication among the many authorities of the businesses, the plan's execution is inefficient, and the plan is doomed to failure.**
- **Despite the fact that financial plans are founded on assumption, there are occasions when the assumptions result in incorrect forecasts.**
- **Some financial plans are inflexible, and they do not change in response to changes in the economic climate or changes in government policy.**
- **Because of the manual data entry, the human error rate of the app remains high.**
- **Manual data entry also makes the entire procedure time-consuming.**

5:APPLICATION:

- **Money matters have a wide range of applications in our daily lives, businesses, and society as a whole. Here are some examples:**

- **Personal finance: Money matters are crucial for managing personal finances. Budgeting, saving, investing, and borrowing all involve making smart financial decisions that can impact our future financial stability and wellbeing.**

- **Banking and financial services: Money matters are at the core of banking and financial services. Financial institutions provide a variety of services such as checking and savings accounts, loans, mortgages, credit cards, and investment products, all of which involve managing money.**

- **Business and entrepreneurship: Money matters are critical for businesses and entrepreneurs. Managing cash flow, raising capital, making investment decisions, and managing risks all require sound financial management.**

- **Government and public policy: Money matters are essential for governments to manage their budgets, fund public services, and drive economic growth. Public policy decisions such as taxation, fiscal policy, and monetary policy can have significant impacts on individuals and businesses.**

- **Philanthropy and social impact: Money matters play a critical role in philanthropy and social impact investing. Philanthropic organizations and impact investors use their money to drive positive social and environmental outcomes, such as supporting education, healthcare, and sustainable development.**

- **Overall, money matters are a fundamental aspect of our lives and society, and understanding how to manage and invest money effectively can have a significant impact on our financial wellbeing and quality of life**

- **The app development cost for this type of app is high compared to a simple financial app.**

6:CONCLUSION:

As an AI language model, I cannot predict the future of money matters with certainty. However, based on current trends and expert analysis, we can make some educated guesses about what the future might hold.

One trend that is likely to continue is the shift towards digital payments and cryptocurrencies. Digital payments are becoming increasingly popular due to their convenience and speed, and many countries are exploring the use of digital currencies or central bank digital currencies (CBDCs) as a way to improve financial inclusion and reduce transaction costs. Cryptocurrencies like Bitcoin and Ethereum are also gaining acceptance as a store of value and a means of exchange, although their volatility and regulatory uncertainty remain concerns for some.

Another trend is the increasing use of artificial intelligence and machine learning in financial services. AI is already being used to automate tasks like fraud detection, credit scoring, and investment management, and is expected to play an even larger role in the future. AI-powered chatbots and virtual assistants are also becoming more common in banking and financial services, providing customers with personalized advice and assistance.

Finally, there is growing interest in sustainable finance and impact investing, as investors seek to align their portfolios with their values and address global challenges like climate change and inequality. This trend is expected to continue as awareness of environmental, social, and governance (ESG) factors increases and more investors demand transparency and accountability from companies and financial institutions.

Overall, the future of money matters is likely to be shaped by digitalization, innovation, and changing social and environmental priorities.

7:FUTURE SCOPES:

Scope of Financial Management
Financial management encompasses four major areas:

- ### Planning

  The financial manager projects how much money the company will need in order to maintain positive cash flow, allocate funds to grow or add new products or services and cope with unexpected events, and shares that information with business colleagues.

- Planning may be broken down into categories including capital expenses, T&E and workforce and indirect and operational expenses.

- ### Budgeting

  The financial manager allocates the company's available funds to meet costs, such as mortgages or rents, salaries, raw materials, employee T&E and other obligations. Ideally there will be some left to put aside for emergencies and to fund new business opportunities.

- Companies generally have a master budget and may have separate sub documents covering, for example, cash flow and operations; budgets may be static or flexible.

- ## Flexible

- Remains the same even if there are significant changes from the assumptions made during planning.

- Adjusts based on changes in the assumptions used in the planning process.

- ## Managing and assessing risk

  Line-of-business executives look to their financial managers to assess and provide compensating controls for a variety of risks, including:

- ## Market risk

Affects the business' investments as well as, for public companies, reporting and stock performance. May also reflect financial risk particular to the industry, such as a pandemic affecting restaurants or the shift of retail to a direct-to-consumer model.

- ## Credit risk

The effects of, for example, customers not paying their invoices on time and thus the business not having funds to meet obligations, which may adversely affect creditworthiness and valuation, which dictates ability to borrow at favorable rates.

- ## Liquidity risk

Finance teams must track current cash flow, estimate future cash needs and be prepared to free up working capital as needed.

- ## Operational risk

This is a catch-all category, and one new to some finance teams. It may include, for example, the risk of a cyber-attack and whether to purchase cybersecurity insurance, what disaster recovery and business continuity plans are in place and what crisis management practices are triggered if a senior executive is accused of fraud or misconduct.

Key Features :

1. Registration And Onboarding

App Onboarding UI

The statement – “The simpler, the better” holds for any mobile app, more so for a financial app. However, most financial apps are complex. Therefore, you must work out all the details to make your users comfortable with your solution.

Let’s take the registration process. It is crucial to make the process of registration smooth.

A lot of data needs to be verified in a personal finance app. You can decide to split the registration form into multiple steps. No more than two to three questions should be asked at every step.

In addition, to enhance security, including biometric authentication here.

2. Account Integration

Your budget management software should be able to gather all financial accounts, such as credit cards, debit cards, loans, bank accounts, and mutual funds of the app user.

In short, your app should be a one-stop solution for everything about money management.

3. Insightful Advice via AI Chatbots

Chatbot

AI or Artificial Intelligence in fintech can vastly increase your team’s productivity, even surpass humans in certain aspects.

Because customers look for conciseness, accuracy, and responsiveness, especially in finance-related issues, AI chatbots prove beneficial here.

AI allows chatbots to analyze user behavior. As a result, they bring a lot of value to the users by giving them insightful advice and desirable information in no time, turning them into a financial coach.

4. Real-time Spending Tracker

A money management app is incomplete without a real-time expenditure tracking feature. Such a feature comes in handy as it saves time and money for users.

You should not need your consumers to switch apps to track where they are spending/investing their money. This should be handled on a single platform.

5. Personalization

By customizing the user experience in your app as per the user's requirements, your personal finance app can become more appealing to the user. For example, display the user's transaction history in an interestingly designed infographic or customized reports. You can also create a helpful chatbot for the user.

6. Gamification

Gamification is a fascinating concept. App developers across many industries are using it these days. It's a wonderful way of keeping your users engaged.

By gamifying the user experience, you encourage users to interact with the application more frequently. On the other hand, the user considers a gamified money app a fun way to help them achieve their saving goals better.

By employing the gamification concept in your personal finance app, you can

Incentivize your users to save more
Improve user's daily financial management
Enhance user's knowledge of financial services and products

7. 24x7 Customer Support

Constant customer support in a financial app is of great importance. Customer support in multiple languages can get you clientele from across the world and keep your customers happy and loyal.

8. Alerts and Notifications

Mobile App Notifications

This is one of the most critical features included in a finance app. Users in case of need notifications and alerts

- *More spendings*
- *Low balance in the account (wallets or bank)*
- *Upcoming bills*
- *Interesting investment deals*

9. Points and Rewards

The app users need motivation from time to time. That's why you must keep offering them rewards and points.

They can be in the form of discounts, cashback, coupons, and credits.

10. Financial Advice & Consultation

Make sure your app also provides financial suggestions and advice from experts.

This would benefit users as they'll learn about additional ways to save money, invest, and keep track of their expenses.

In addition, you can include the following advanced features in your personal finance management solution.

- ***Common currency converter***
- ***Calculator***
- ***Credit score calculator***
- ***Tax calculator***

• Procedures

The financial manager sets procedures regarding how the finance team will process and distribute financial data, like invoices, payments and reports, with security and accuracy. These written procedures also outline who is responsible for making financial decisions at the company — and who signs off on those decisions.

Budget
Plan
Track
Report Costs
Analyze
Managing Personal Finance

8:APPENDIX:

Source Code;

1.Creating the database classes for user login and registration

Step 1 : Create User data class

Packagecom.example.expensestracker

```
Importandroidx.room.ColumnInfo

Importandroidx.room.Entity

Importandroidx.room.PrimaryKey


@Entity(tableName =”user_table”)

Data classUser(

   @PrimaryKey(autoGenerate =true) valid:Int?,

   @ColumnInfo(name =”first_name”) valfirstName:String?,

   @ColumnInfo(name =”last_name”) vallastName:String?,

   @ColumnInfo(name =”email”) valemail:String?,

   @ColumnInfo(name =”password”) valpassword:String?,


   )
```

Step 2 : Create an UserDao interface

```
Packagecom.example.expensestracker


Importandroidx.room.*
```

```
@Dao
interfaceUserDao{

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

@Insert(onConflict =OnConflictStrategy.REPLACE)
suspendfuninsertUser(user:User)

@Update
suspendfunupdateUser(user:User)

@Delete
suspendfundeleteUser(user:User)
}
```

Step 3 : Create an UserDatabase class

```
Packagecom.example.expensestracker
```

```
Importandroid.content.Context
Importandroidx.room.Database
Importandroidx.room.Room
Importandroidx.room.RoomDatabase

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

  abstractfunuserDao(): UserDao

  companionobject{

    @Volatile
    Privatevarinstance:UserDatabase?=null

    fungetDatabase(context:Context): UserDatabase{
      returninstance ?:synchronized(this) {
        valnewInstance =Room.databaseBuilder(
          context.applicationContext,
            UserDatabase::class.java,
            “user_database”
          ).build()
          Instance =newInstance
          newInstance
        }
      }
    }
  }
```

Step 4 : Create an UserDatabaseHelper class

```
Packagecom.example.expensestracker

Importandroid.annotation.SuppressLint
Importandroid.content.ContentValues
Importandroid.content.Context
Importandroid.database.Cursor
Importandroid.database.sqlite.SQLiteDatabase
```

```
 Importandroid.database.sqlite.SQLiteOpenHelper


 classUserDatabaseHelper(context:Context) :
    SQLiteOpenHelper(context, DATABASE_NAME, null,
DATABASE_VERSION) {

    Companionobject{
       privateconstvalDATABASE_VERSION=1
       privateconstvalDATABASE_NAME=”UserDatabase.db”

       privateconstvalTABLE_NAME=”user_table”
       privateconstvalCOLUMN_ID=”id”
       privateconstvalCOLUMN_FIRST_NAME=”first_name”
       privateconstvalCOLUMN_LAST_NAME=”last_name”
       privateconstvalCOLUMN_EMAIL=”email”
       privateconstvalCOLUMN_PASSWORD=”password”
    }

    overridefunonCreate(db:SQLiteDatabase?) {
       valcreateTable =”CREATE TABLE $TABLE_NAME(“+
            “$COLUMN_IDINTEGER PRIMARY KEY
AUTOINCREMENT, “+
            “$COLUMN_FIRST_NAMETEXT, “+
            “$COLUMN_LAST_NAMETEXT, “+
            “$COLUMN_EMAILTEXT, “+
            “$COLUMN_PASSWORDTEXT”+
            “)”

      Db?.execSQL(createTable)
    }

    overridefunonUpgrade(db:SQLiteDatabase?,
oldVersion:Int, newVersion:Int) {
      db?.execSQL(“DROP TABLE IF EXISTS $TABLE_NAME”)
      onCreate(db)
    }

    funinsertUser(user:User) {
      valdb =writableDatabase
      valvalues =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")
    fungetUserByUsername(username:String): User?{
        valdb =readableDatabase
        valcursor:Cursor=db.rawQuery("SELECT * FROM
$TABLE_NAMEWHERE $COLUMN_FIRST_NAME= ?",
arrayOf(username))
        varuser:User?=null
        if(cursor.moveToFirst()) {
            user =User(
                id
=cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                firstName
=cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NA
ME)),
            lastName
=cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NA
ME)),
            email
=cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
            password
=cursor.getString(cursor.getColumnIndex(COLUMN_PASSWOR
D)),
          )
        }
        Cursor.close()
        Db.close()
        Returnuser
    }
    @SuppressLint("Range")
    fungetUserById(id:Int): User?{
        valdb =readableDatabase
        valcursor:Cursor=db.rawQuery("SELECT * FROM
$TABLE_NAMEWHERE $COLUMN_ID= ?",
arrayOf(id.toString()))
        varuser:User?=null
        if(cursor.moveToFirst()) {
           user =User(
              id
=cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
              firstName
=cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NA
ME)),
              lastName
=cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NA
ME)),
              email
=cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
              password
=cursor.getString(cursor.getColumnIndex(COLUMN_PASSWOR
D)),
           )
        }
        Cursor.close()
        Db.close()
        Returnuser
     }

     @SuppressLint(“Range”)
```

```
    fungetAllUsers(): List<User> {
        valusers =mutableListOf<User>()
        valdb =readableDatabase
        valcursor:Cursor=db.rawQuery(“SELECT * FROM
$TABLE_NAME”, null)
        if(cursor.moveToFirst()) {
            do{
                valuser =User(
                    id
=cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                    firstName
=cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NA
ME)),
                    lastName
=cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NA
ME)),
                    email
=cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                    password
=cursor.getString(cursor.getColumnIndex(COLUMN_PASSWOR
D)),
                )
                Users.add(user)
            } while(cursor.moveToNext())
        }
        Cursor.close()
        Db.close()
        Returnusers
    }

}
```

2.Creating the database classes for item name, quantity and cost.

Step 1 : Create Items data class

```
Packagecom.example.expensestracker

Importandroidx.room.ColumnInfo
Importandroidx.room.Entity
Importandroidx.room.PrimaryKey

@Entity(tableName =”items_table”)
Data classItems(
   @PrimaryKey(autoGenerate =true) valid:Int?,
   @ColumnInfo(name =”item_name”) valitemName:String?,
   @ColumnInfo(name =”quantity”) valquantity:String?,
   @ColumnInfo(name =”cost”) valcost:String?,
)
```

Step 2 : Create ItemsDao interface

```
Packagecom.example.expensestracker

Importandroidx.room.*

@Dao
interfaceItemsDao{
    @Query(“SELECT * FROM items_table WHERE  cost= :cost”)
    suspendfungetItemsByCost(cost:String): Items?

    @Insert(onConflict =OnConflictStrategy.REPLACE)
    suspendfuninsertItems(items:Items)

    @Update
    suspendfunupdateItems(items:Items)

    @Delete
    suspendfundeleteItems(items:Items)
}
```

Step 3 : Create ItemsDatabse class

```
Packagecom.example.expensestracker

Importandroid.content.Context
```

```
Importandroidx.room.Database
Importandroidx.room.Room
Importandroidx.room.RoomDatabase

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

    abstractfunItemsDao(): ItemsDao

    companionobject{

        @Volatile
        Privatevarinstance:ItemsDatabase?=null

        fungetDatabase(context:Context): ItemsDatabase{
            returninstance ?:synchronized(this) {
                valnewInstance =Room.databaseBuilder(
                    context.applicationContext,
                    ItemsDatabase::class.java,
                    "items_database"
            ).build()
            Instance =newInstance
            newInstance
         }
      }
   }
}
```

Step 4 : Create ItemsDatabaseHelper class

```
Packagecom.example.expensestracker

Importandroid.annotation.SuppressLint
Importandroid.content.ContentValues
Importandroid.content.Context
Importandroid.database.Cursor
Importandroid.database.sqlite.SQLiteDatabase
Importandroid.database.sqlite.SQLiteOpenHelper
```

```
  classItemsDatabaseHelper(context:Context) :
     SQLiteOpenHelper(context, DATABASE_NAME,
null,DATABASE_VERSION){

     Companionobject{
        privateconstvalDATABASE_VERSION=1
        privateconstvalDATABASE_NAME=”ItemsDatabase.db”

        privateconstvalTABLE_NAME=”items_table”
        privateconstvalCOLUMN_ID=”id”
        privateconstvalCOLUMN_ITEM_NAME=”item_name”
        privateconstvalCOLUMN_QUANTITY=”quantity”
        privateconstvalCOLUMN_COST=”cost”
     }

     overridefunonCreate(db:SQLiteDatabase?) {
        valcreateTable =”CREATE TABLE $TABLE_NAME(“+
              “${COLUMN_ID}INTEGER PRIMARY KEY
AUTOINCREMENT, “+
              “${COLUMN_ITEM_NAME}TEXT,”+
            “${COLUMN_QUANTITY}TEXT,”+
            “${COLUMN_COST}TEXT”+
            “)”

        Db?.execSQL(createTable)
    }

    overridefunonUpgrade(db:SQLiteDatabase?,
oldVersion:Int, newVersion:Int) {
        db?.execSQL(“DROP TABLE IF EXISTS $TABLE_NAME”)
        onCreate(db)
    }

    funinsertItems(items:Items) {
        valdb =writableDatabase
        valvalues =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")
    fungetItemsByCost(cost:String): Items?{
        valdb =readableDatabase
        valcursor:Cursor=db.rawQuery("SELECT * FROM
$TABLE_NAMEWHERE $COLUMN_COST= ?", arrayOf(cost))
        varitems:Items?=null
        if(cursor.moveToFirst()) {
            items =Items(
                id
=cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                itemName
=cursor.getString(cursor.getColumnIndex(COLUMN_ITEM_NA
ME)),
                quantity
=cursor.getString(cursor.getColumnIndex(COLUMN_QUANTIT
Y)),
              cost
=cursor.getString(cursor.getColumnIndex(COLUMN_COST)),
           )
        }
        Cursor.close()
        Db.close()
        Returnitems
    }
    @SuppressLint("Range")
    fungetItemsById(id:Int): Items?{
        valdb =readableDatabase
        valcursor:Cursor=db.rawQuery("SELECT * FROM
$TABLE_NAMEWHERE $COLUMN_ID= ?",
arrayOf(id.toString()))
        varitems:Items?=null
        if(cursor.moveToFirst()) {
            items =Items(
              id
=cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
              itemName
=cursor.getString(cursor.getColumnIndex(COLUMN_ITEM_NA
ME)),
            quantity
=cursor.getString(cursor.getColumnIndex(COLUMN_QUANTIT
Y)),
            cost
=cursor.getString(cursor.getColumnIndex(COLUMN_COST)),
         )
       }
       Cursor.close()
       Db.close()
       Returnitems
    }

    @SuppressLint("Range")
    fungetAllItems(): List<Items> {
       valitem =mutableListOf<Items>()
       valdb =readableDatabase
       valcursor:Cursor=db.rawQuery("SELECT * FROM
$TABLE_NAME", null)
       if(cursor.moveToFirst()) {
          do{
             valitems =Items(
                    id
=cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                    itemName
=cursor.getString(cursor.getColumnIndex(COLUMN_ITEM_NA
ME)),
                    quantity
=cursor.getString(cursor.getColumnIndex(COLUMN_QUANTIT
Y)),
                    cost
=cursor.getString(cursor.getColumnIndex(COLUMN_COST)),
                )
                Item.add(items)
            } while(cursor.moveToNext())
        }
        Cursor.close()
        Db.close()
        Returnitem
    }
 }
```