Project Report Template

1. INTRODUCTION

Money management is a critical aspect of personal finance that involves the process of planning, organizing, and controlling one's financial resources effectively. It is an essential life skill that can help individuals achieve financial stability, build wealth, and attain their long-term financial goals.Effective money management begins with creating a budget. A budget is a plan that outlines an individual's expected income and expenses over a period. It helps individuals track their spending, identify areas where they can cut back, and save for future expenses. To create a budget, one needs to determine their monthly income, fixed expenses such as rent, utilities, and loan repayments, and variable expenses such as groceries, entertainment, and transportation. With this information, one can create a budget that allocates their income appropriately, allowing them to prioritize their expenses and save for future goals.

Another important aspect of money management is debt management. Many people face debt in various forms, such as credit card debt, student loans, or mortgages. Effective debt management involves creating a plan to pay off debt systematically, starting with the debt with the highest interest rate. This approach can help individuals save money in interest charges over time and reduce their overall debt burden. Savings and investment are also crucial components of money management. Saving money involves setting aside a portion of one's income for future use, such as an emergency fund or a down payment for a home. Investment, on the other hand, involves putting money into assets such as stocks, bonds, or real estate with the aim of earning a return on investment. Individuals need to assess their risk tolerance, financial goals, and investment options before deciding on the best investment strategy. Effective money management also involves being mindful of one's spending habits. It is essential to differentiate between needs and wants and to prioritize essential expenses over discretionary ones. It is also crucial to be aware of one's behavioral biases that may affect financial decisions, such as impulse buying or herd mentality. Being mindful of these biases can help individuals make informed financial decisions that align with their long-term goals.

Finally, it is essential to be financially literate to make informed financial decisions. Financial literacy involves understanding financial concepts such as budgeting, debt management, savings, and investment. It also involves being aware of financial regulations, consumer rights, and financial fraud. Individuals can improve their financial literacy by reading books, attending financial education workshops, or seeking advice from financial professionals.

In conclusion, effective money management is a critical life skill that can help individuals achieve financial stability and attain their long-term financial goals. It involves creating a budget, managing debt, saving and investing wisely, being mindful of one's spending habits, and improving financial literacy. By practicing good money management habits, individuals can take control of their finances and live a financially secure life.

1.1 overview

Money management, or personal finance management, refers to the process of planning, organizing, and controlling one's financial resources effectively. It is a critical life skill that can help individuals achieve financial stability, build wealth, and attain their long-term financial goals.There are several key components to effective money management, including:

Budgeting: Creating a budget is the foundation of effective money management. A budget is a plan that outlines an individual's expected income and expenses over a period. It helps individuals track their spending, identify areas where they can cut back, and save for future expenses.

Debt Management: Debt management involves creating a plan to pay off debt systematically, starting with the debt with the highest interest rate. This approach can help individuals save money in interest charges over time and reduce their overall debt burden.

Savings and Investment: Saving money involves setting aside a portion of one's income for future use, such as an emergency fund or a down payment for a home. Investment involves putting money into assets such as stocks, bonds, or real estate with the aim of earning a return on investment.

Spending Habits: Being mindful of one's spending habits is essential to effective money management. It is essential to differentiate between needs and wants and to prioritize essential expenses over discretionary ones.

Financial Literacy: Financial literacy involves understanding financial concepts such as budgeting, debt management, savings, and investment. It also involves being aware of financial regulations, consumer rights, and financial fraud.

By practicing good money management habits, individuals can take control of their finances and live a financially secure life.

1.2 purposes

Personal finance management involves the process of managing one's financial resources, which includes budgeting, saving, investing, and managing debt. In this article, we will discuss the purposes of money matters in personal finance management.Meeting Basic Needs: The primary purpose of money is to meet our basic needs, such as food, shelter, and clothing. These necessities are essential for survival and require a steady flow of income to maintain. Personal finance management involves creating a budget that allocates a sufficient amount of money to cover basic expenses.

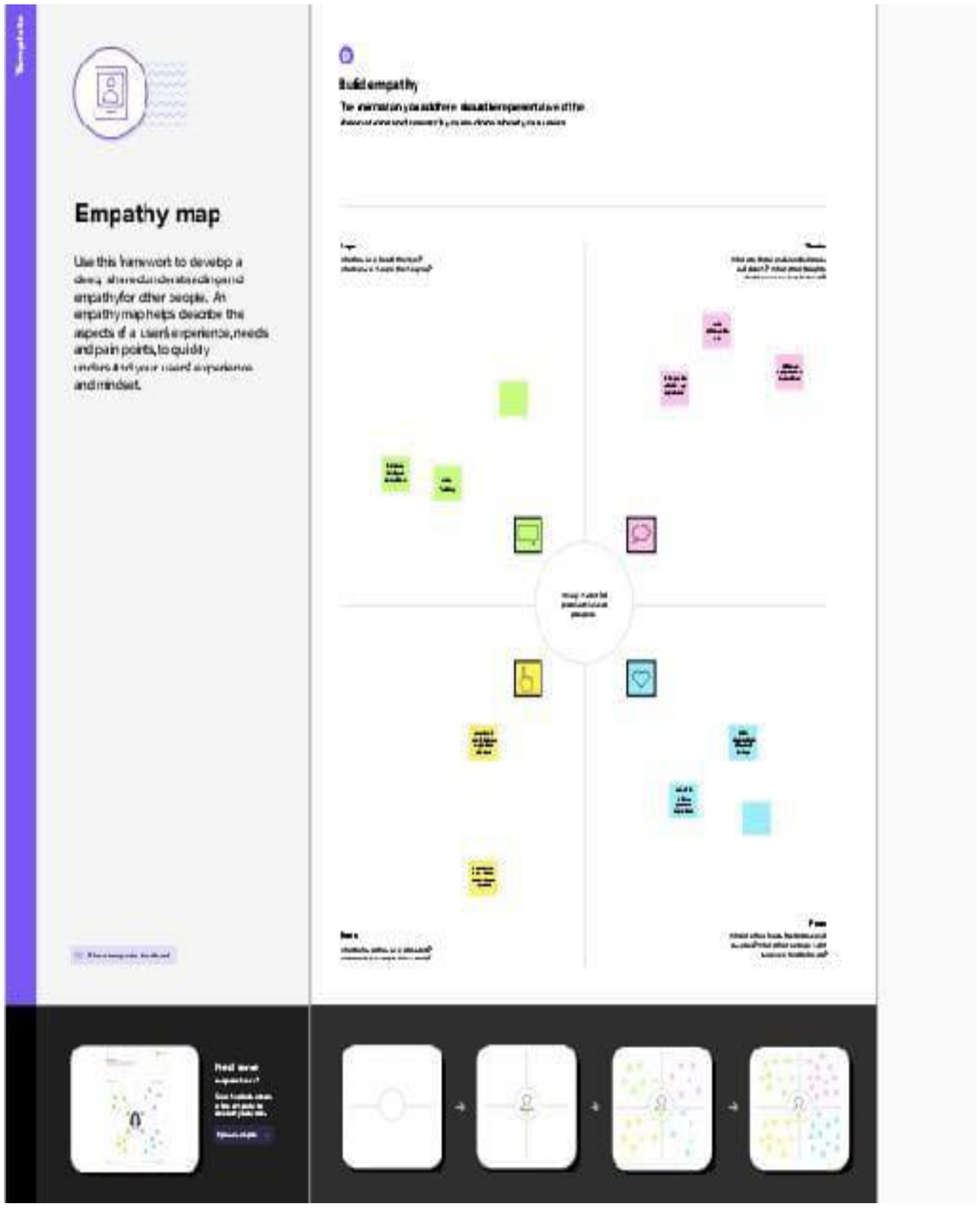
Managing Financial Emergencies: Unexpected events, such as job loss, medical emergencies, or car repairs, can have a significant impact on our finances. Having money set aside for emergencies can help us avoid financial hardships and reduce stress. Personal finance management involves creating an emergency fund that covers at least six months of living expenses.

Saving for the Future: Money can help us achieve our long-term financial goals, such as buying a house, starting a business, or retirement. Saving money and investing in assets that appreciate over time can help us build wealth and achieve financial independence. Personal finance management involves creating a savings plan and investing in assets that align with our long-term goals.

Managing Debt: Debt can be a useful tool for achieving our financial goals, such as buying a house or financing education. However, too much debt can be a burden and lead to financial stress. Personal finance management involves managing debt by creating a debt repayment plan and avoiding taking on too much debt.

Achieving Financial Freedom: Money can provide us with the freedom to live the life we want. It can allow us to travel, pursue our passions, or start a business. Personal finance management involves creating a financial plan that aligns with our values and goals and provides us with the financial freedom to pursue our dreams.

2. PROJECT DEFINITION AND DESIGN THINKING

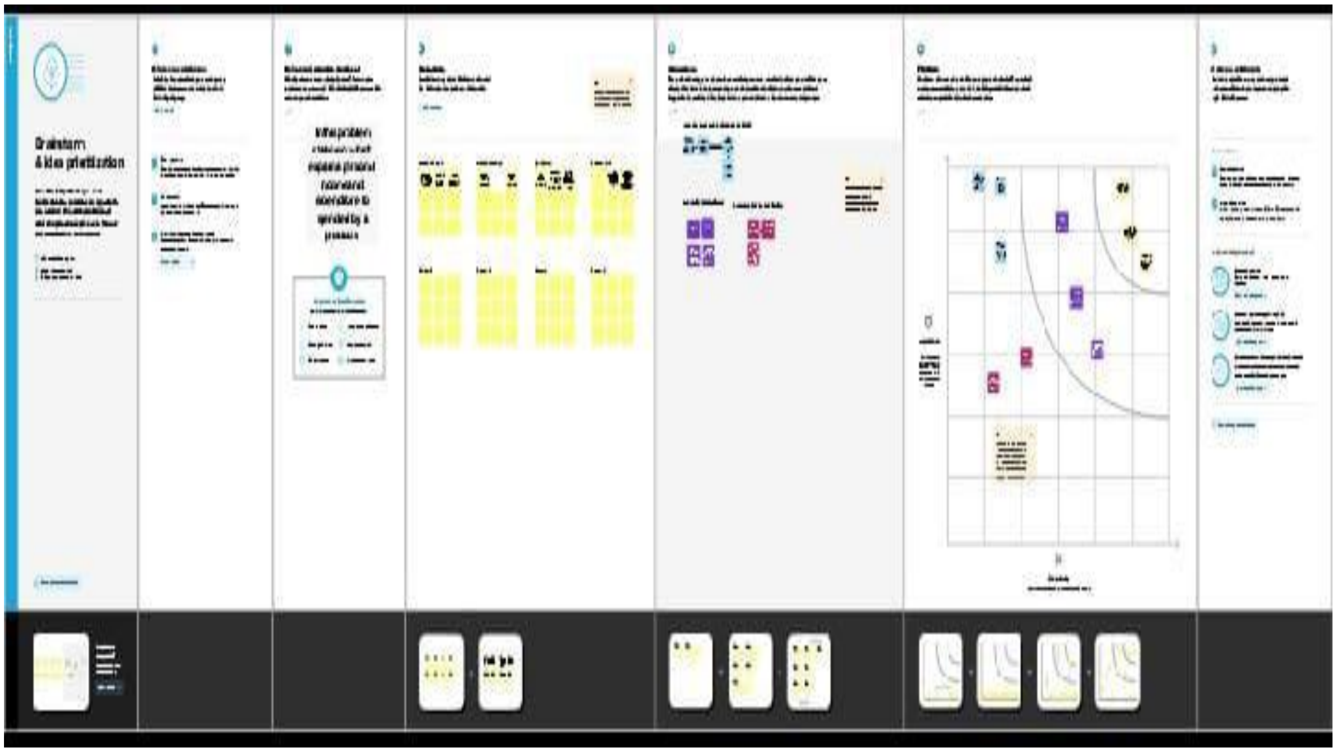


Empathy map

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

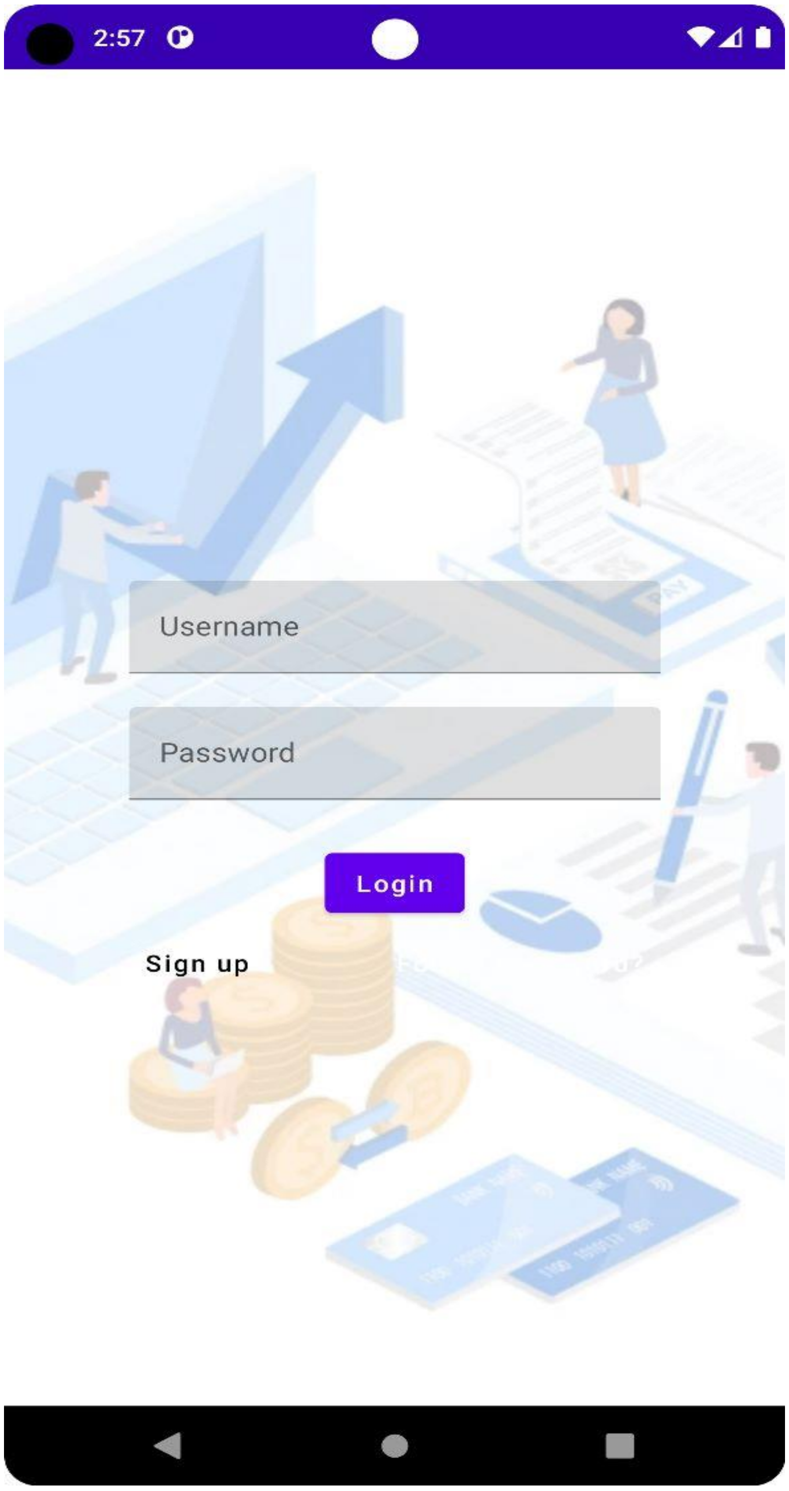
Build empathy

2.1 Empathy Map

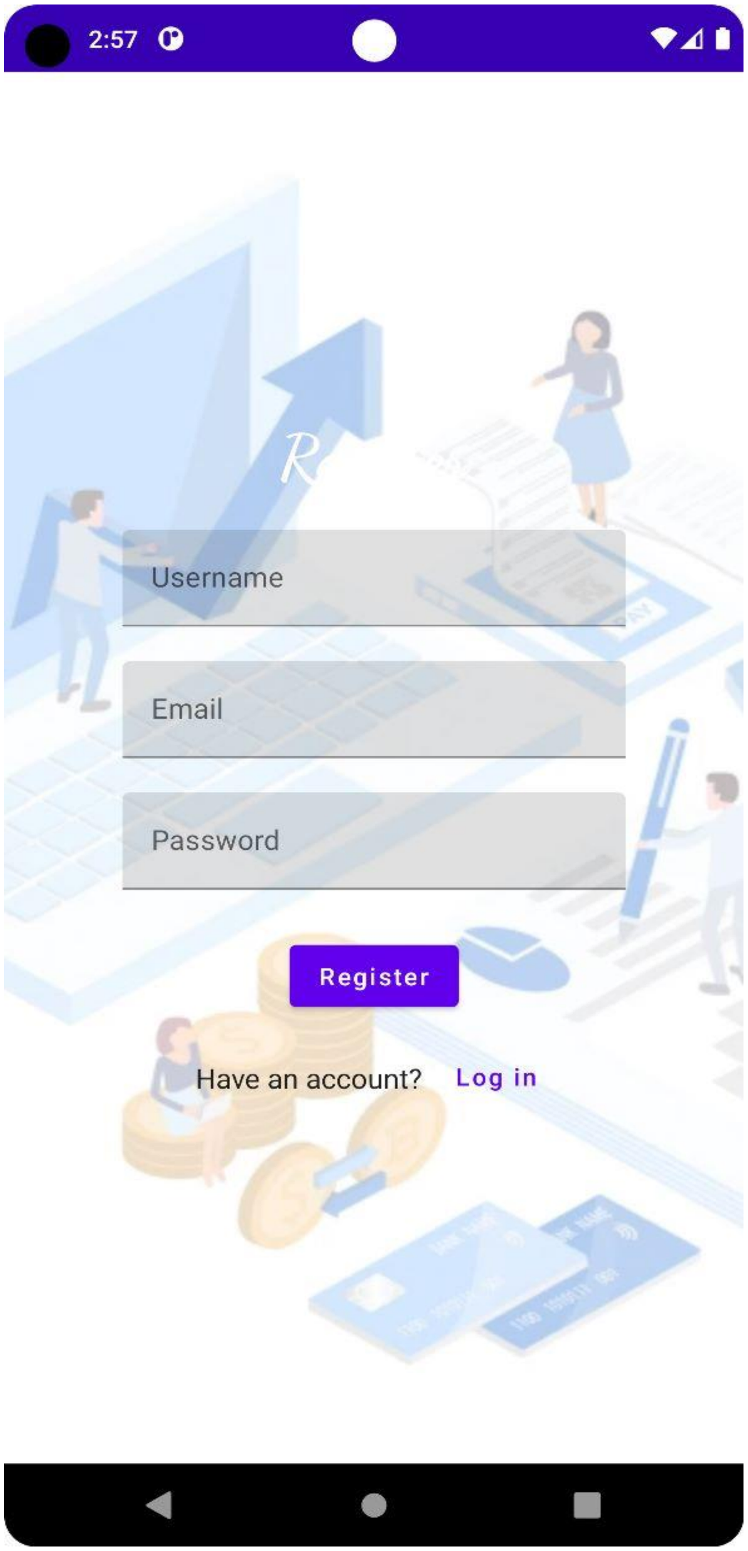


2.2 Ideation & Brainstorming Map

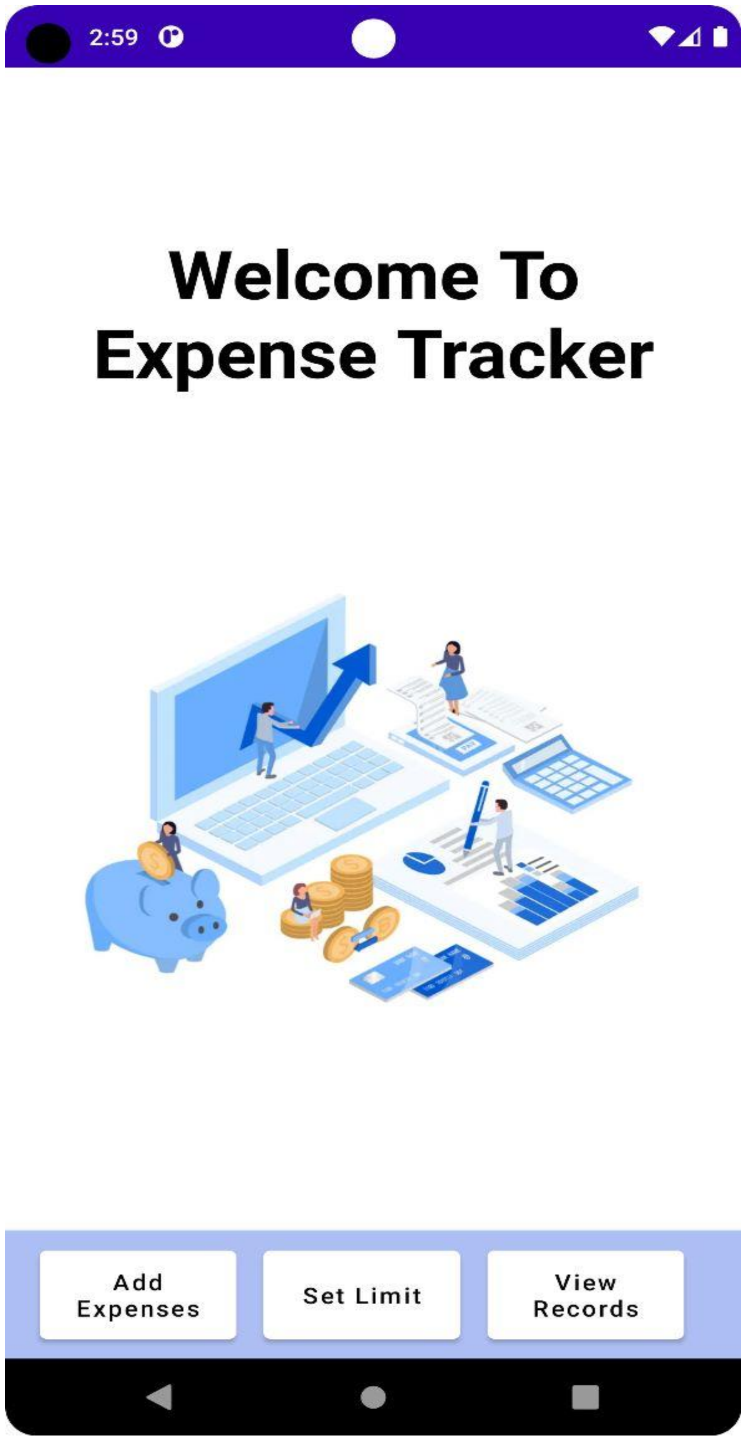
3. RESULT



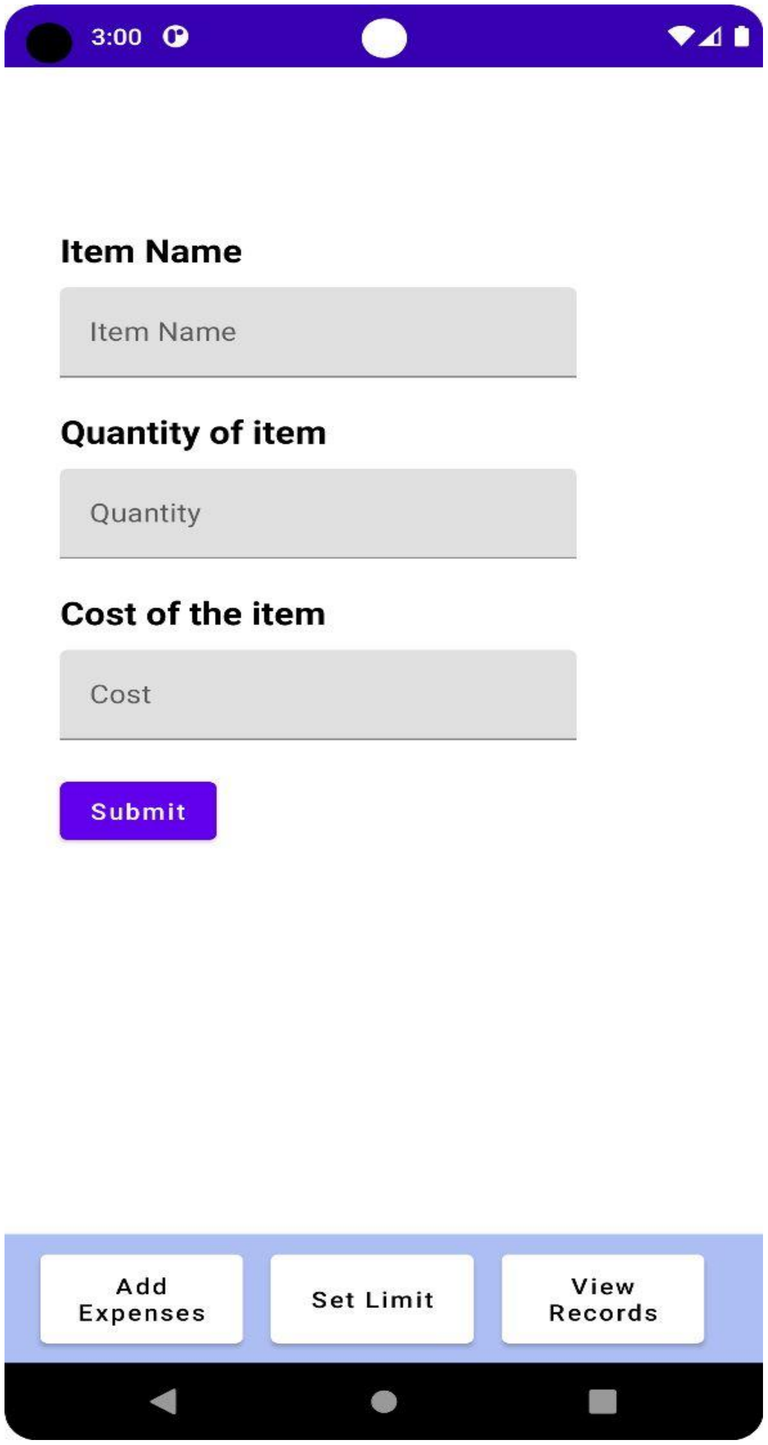
3.1 Above activity mention login page of project



3.2 This above activity shows a registration page



3.3 Above mentioned activity intro page


3:00

Item Name

Item Name

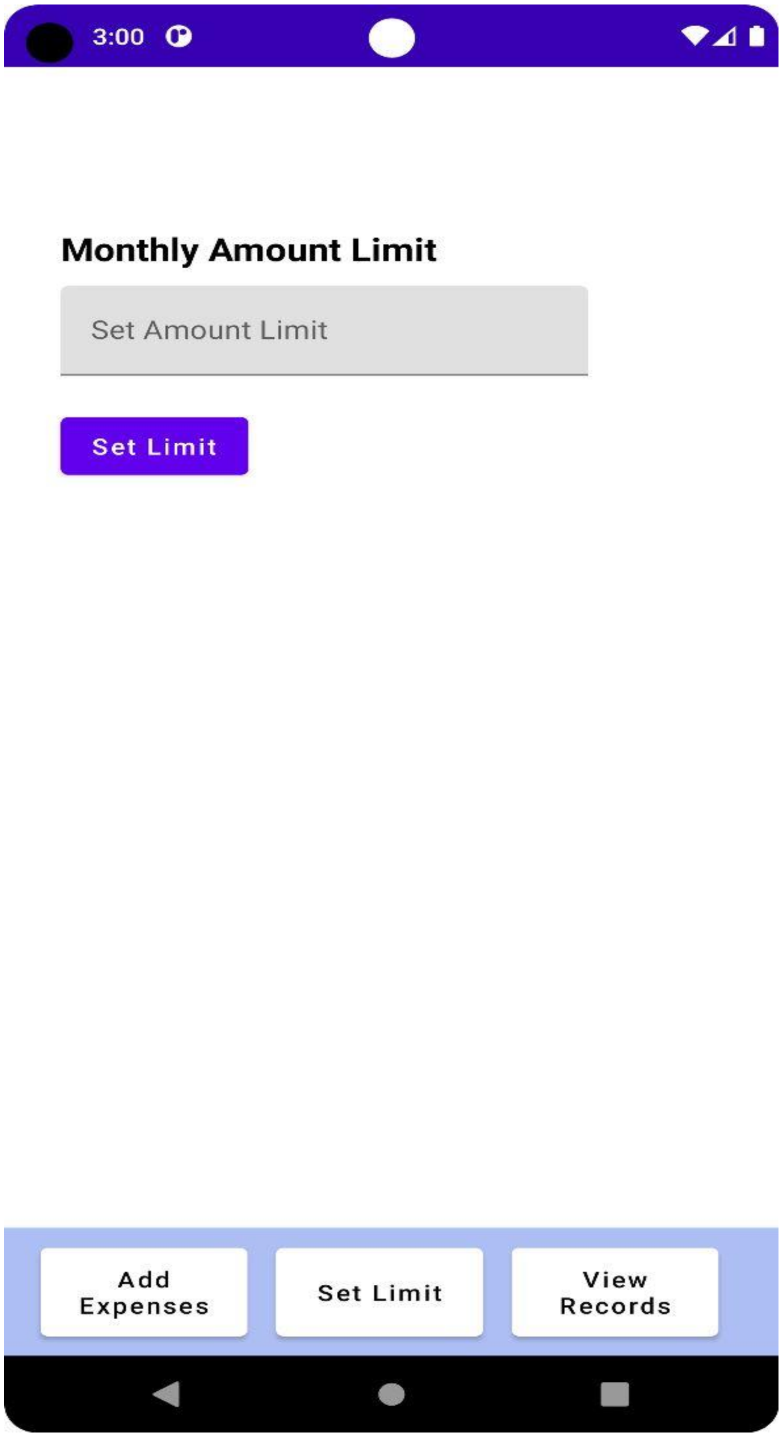
Quantity of item

Quantity

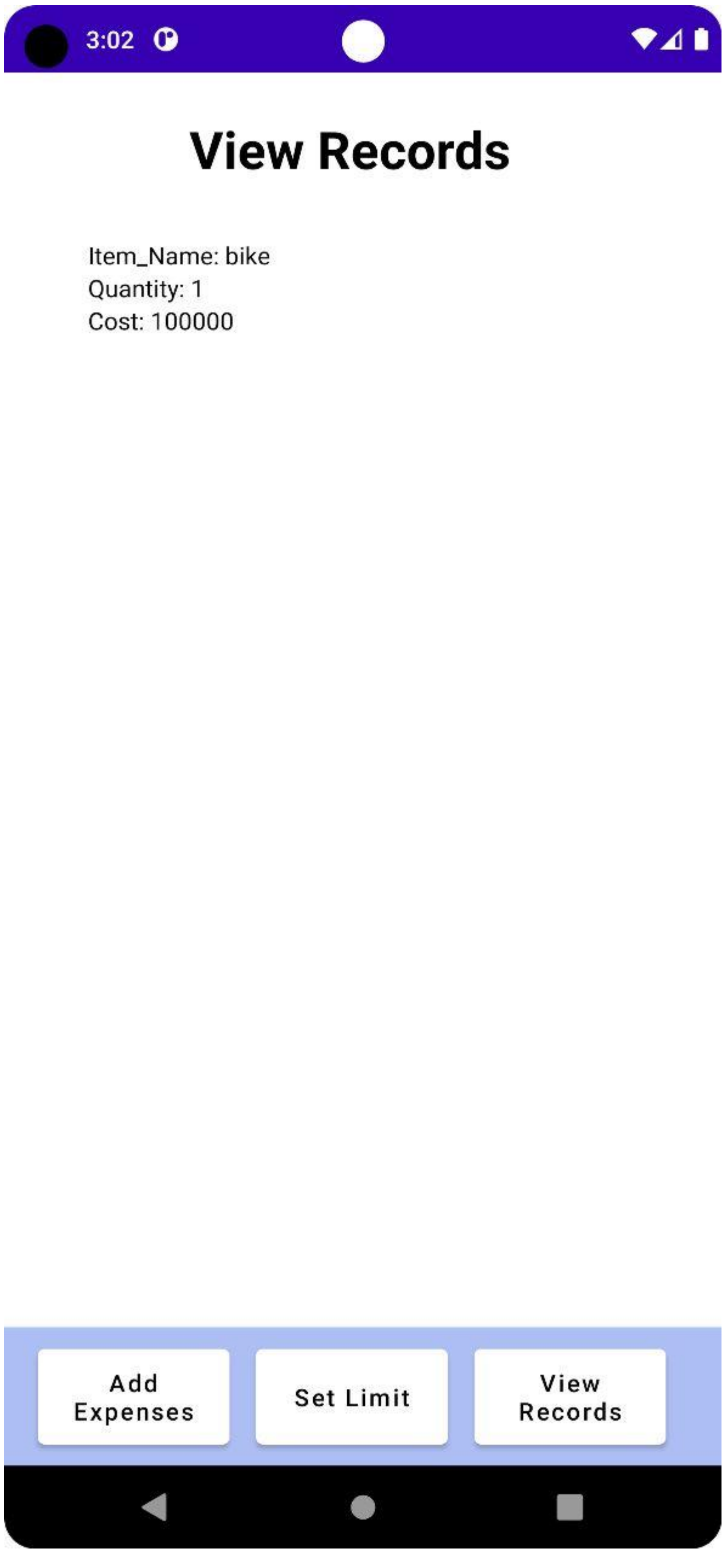
Cost of the item

Cost

3.4 Above mentioned activity describe a list of item with price benefits



3.5 Above activity shows out price limit and strategy



View Records

Item_Name: bike
Quantity: 1
Cost: 100000

3.6 Above activity mention given outsources

4. ADVANTAGES & DISADVANTAGES

Personal finance management refers to the process of managing one's financial resources effectively, including budgeting, saving, investing, and managing debt. Effective personal finance management can bring numerous benefits, but it also has its challenges and drawbacks. In this essay, we will discuss the advantages and disadvantages of personal finance management.

Advantages:

Improved Financial Stability: One of the most significant advantages of personal finance management is improved financial stability. By managing their finances effectively, individuals can create a budget, save for emergencies and future expenses, and pay off debt. This can help them achieve financial stability, reduce financial stress, and avoid living paycheck to paycheck.Increased Savings: Personal finance management can help individuals save money for various goals, such as buying a house, starting a business, or retiring. By creating a budget and reducing unnecessary expenses, individuals can save more money and make progress toward their financial goals.

Better Investment Decisions: Effective personal finance management can also help individuals make better investment decisions. By understanding their risk tolerance and investment options, individuals can choose the right investment vehicles and create a diversified investment portfolio that aligns with their goals.

Reduced Debt: Personal finance management can also help individuals reduce debt and manage their debt effectively. By creating a debt payoff plan and prioritizing high-interest debt, individuals can reduce their debt load and save money on interest payments.

Improved Credit Score: Managing one's finances effectively can also lead to an improved credit score. By paying bills on time, reducing debt, and using credit responsibly, individuals can improve their creditworthiness and qualify for better credit terms and lower interest rates.

Disadvantages:

Time and Effort: Personal finance management requires time and effort to create a budget, track expenses, and make financial decisions. This can be challenging for individuals with busy schedules or those who lack financial knowledge or skills.

Financial Stress: Although personal finance management can reduce financial stress in the long run, it can also cause stress in the short term. For example, creating a budget and cutting expenses can be difficult and require sacrifices.

Risky Investments: Personal finance management can lead to better investment decisions, but it can also lead to risky investments. Individuals with limited financial knowledge or experience may make poor investment decisions or fall victim to scams or fraudulent schemes.

Opportunity Cost: Personal finance management requires individuals to prioritize their financial goals and make trade-offs. This means that individuals may miss out on certain opportunities, such as travel or entertainment, in order to save money or invest for the future.

Inflexibility: Personal finance management can also be inflexible, especially if individuals have limited income or face unexpected expenses. This can make it difficult to stick to a budget or save money consistently.

Conclusion:

In conclusion, personal finance management has numerous advantages and disadvantages. While it can improve financial stability, increase savings, and lead to better investment decisions, it also requires time and effort, can cause financial stress, and may lead to risky investments. It is essential for individuals to weigh these pros and cons carefully and make informed decisions that align with their financial goals and values. With the right approach and mindset, personal finance management can be a powerful tool for achieving financial security and success.

5. APPLICATIONS

Personal finance management applications and solutions can be incredibly useful for individuals who want to keep track of their finances and improve their financial well-being. There are several different ways that these applications can be used to help manage personal finances, and in this article, we will explore some of the key benefits and use cases of personal finance management applications.

Budgeting: One of the most common uses of personal finance management applications is for budgeting. With these tools, individuals can create budgets for various expenses, such as rent, utilities, groceries, entertainment, and more. Users can set spending limits for each category and track their spending throughout the month to ensure they stay within their budget. Budgeting apps can also provide insights and recommendations on how to adjust spending habits to reach financial goals.

Tracking expenses: Personal finance management applications can also be used to track expenses. This is particularly useful for individuals who want to get a better understanding of where their money is going. Users can input all their expenses into the app, either manually or automatically, and see how much they are spending in different categories. By tracking expenses, individuals can identify areas where they may be overspending and make adjustments to their budget accordingly.

Managing debt: Personal finance management applications can also help individuals manage their debt. These apps can show users how much they owe on each debt, including credit cards, loans, and mortgages. Users can set up a payment plan and track their progress towards paying off their debts. Additionally, these apps may provide advice on how to reduce interest rates or negotiate with creditors.

Investing: Personal finance management applications can also be used to help individuals invest their money. These apps can provide insights and recommendations on different investment opportunities, based on the user's financial goals and risk tolerance. Some apps may also offer features that allow users to track their investment portfolios and monitor the performance of their investments.

Saving money: Personal finance management applications can also be used to help individuals save money. These apps can suggest ways to cut expenses or increase income, such as reducing monthly subscriptions, negotiating bills, or finding new job opportunities. Some apps may also offer features that help users set savings goals and track their progress towards achieving those goals.

Tax management: Personal finance management applications can also be used to help individuals manage their taxes. These apps can track income, expenses, and deductions, making it easier to file tax returns. Additionally, some apps may offer tax advice and guidance on how to minimize tax liabilities.

Retirement planning: Personal finance management applications can also be used to help individuals plan for retirement. These apps can provide insights and recommendations on how much to save for retirement, how to invest retirement funds, and when to start taking Social Security benefits. Users can also track their progress towards retirement goals and adjust their plans as necessary.

In conclusion, personal finance management applications can be incredibly useful for individuals who want to improve their financial well-being. These apps can help users budget their expenses, track their spending, manage their debts, invest their money, save money, manage their taxes, and plan for retirement. With the right app and the right approach, individuals can take control of their finances and build a more secure financial future.

6. CONCLUSION

Personal finance management is an essential skill that everyone should acquire. It involves the process of managing one's money, including budgeting, saving, investing, and spending wisely. Good financial management can lead to financial stability, security, and independence. On the other hand, poor financial management can lead to financial stress, debt, and financial difficulties.One of the key aspects of personal finance management is budgeting. A budget is a plan that outlines your income and expenses. It helps you track your spending and ensures that you don't spend more than you earn. Creating a budget involves identifying your sources of income and expenses, setting financial goals, and allocating your resources to meet those goals. A budget can also help you identify areas where you can cut back on expenses and save more money.

Another important aspect of personal finance management is saving. Saving money can help you build a financial safety net, prepare for unexpected expenses, and achieve your long-term financial goals. There are different types of savings, including emergency savings, retirement savings, and short-term savings. Emergency savings are funds set aside to cover unexpected expenses like medical bills or car repairs. Retirement savings are funds set aside for your retirement, while short-term savings are funds set aside for short-term goals like a down payment on a house or a vacation.

Investing is another important aspect of personal finance management. Investing involves using your money to purchase assets that will increase in value over time, such as stocks, bonds, or real estate. Investing can help you grow your wealth and achieve your long-term financial goals. However, investing also comes with risks, and it's important to educate yourself about the risks and benefits of different types of investments before making any investment decisions.

Debt management is another important aspect of personal finance management. Debt can be a useful tool for achieving your financial goals, but it can also lead to financial difficulties if not managed properly. It's important to manage your debt by making timely payments, avoiding high-interest debt, and paying off debt as quickly as possible.

Finally, spending wisely is an important aspect of personal finance management. Spending wisely involves making smart purchasing decisions, avoiding unnecessary expenses, and living within your means. It's important to prioritize your spending based on your financial goals and values and to avoid overspending or making impulse purchases.

In conclusion, personal finance management is an important skill that everyone should acquire. It involves budgeting, saving, investing, debt management, and spending wisely. Good financial management can lead to financial stability, security, and independence, while poor financial management can lead to financial stress, debt, and financial difficulties. By taking control of your finances and making smart financial decisions, you can achieve your financial goals and live a more financially secure and fulfilling life.

7. FUTURE SCOPE

Personal finance management is a crucial aspect of an individual's financial wellbeing. In today's fast-paced world, managing personal finances is not an easy task. The advent of technology has made it easier to manage personal finances, but there is still a lot of scope for improvement. In this article, we will explore some of the future enhancements that can be made in personal finance management.

Artificial Intelligence and Machine Learning

Artificial Intelligence (AI) and Machine Learning (ML) are two technologies that can revolutionize personal finance management. AI and ML can help in analyzing the financial data of an individual and provide personalized financial advice. AI can also help in automating financial tasks, such as bill payments and investments, thereby reducing the workload of individuals.

Blockchain

Blockchain is a decentralized ledger technology that can be used to improve the security of financial transactions. Blockchain technology can be used to create a tamper-proof and transparent system for financial transactions, thereby reducing the risk of fraud and theft.

Personalized Financial Planning

Personalized financial planning is an important aspect of personal finance management. In the future, we can expect to see more personalized financial planning tools that take into account an individual's financial goals, risk tolerance, and investment preferences. These tools can provide customized investment advice, thereby helping individuals to achieve their financial goals.

Robo-Advisors

Robo-advisors are computer programs that provide financial advice based on algorithms. These programs can help individuals to invest their money in a diversified portfolio that matches their risk tolerance and investment goals. Robo-advisors can also help in reducing the fees associated with traditional financial advisors.

Mobile Payments

Mobile payments are becoming increasingly popular as more people are using their mobile phones for financial transactions. In the future, we can expect to see more secure and user-friendly mobile payment systems that offer more features and functionalities.

Financial Education

Financial education is an important aspect of personal finance management. In the future, we can expect to see more online courses and resources that provide financial education to individuals. These resources can help individuals to understand financial concepts and make better financial decisions.

Integration of Financial Data

Integration of financial data is an important aspect of personal finance management. In the future, we can expect to see more integration between different financial platforms, such as banking, investment, and budgeting platforms. This integration can help individuals to get a holistic view of their finances and make better financial decisions.

Cybersecurity

Cybersecurity is a critical aspect of personal finance management. In the future, we can expect to see more secure and advanced cybersecurity measures that protect individuals' financial data from cyber threats.

Tax Management

Tax management is an important aspect of personal finance management. In the future, we can expect to see more advanced tax management tools that help individuals to file their taxes more efficiently and accurately.

Personal Finance Management Apps

Personal finance management apps are becoming increasingly popular as more people are using their mobile phones for financial transactions. In the future, we can expect to see more advanced personal finance management apps that offer more features and functionalities, such as financial planning, investment advice, and budget tracking.

8. APPENDIX

Default ignored files
/shelf/
/workspace.xml
Expenses Tracker

```
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="CompilerConfiguration">
    <bytecodeTargetLevel target="11" />
  </component>
</project>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="GradleSettings">
    <option name="linkedExternalProjectsSettings">
      <GradleProjectSettings>
        <option name="testRunner" value="GRADLE" />
        <option name="distributionType" value="DEFAULT_WRAPPED" />
        <option name="externalProjectPath" value="$PROJECT_DIR$" />
        <option name="modules">
          <set>
            <option value="$PROJECT_DIR$" />
            <option value="$PROJECT_DIR$/app" />
          </set>
        </option>
      </GradleProjectSettings>
    </option>
  </component>
```

```
</project>

<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="ExternalStorageConfigurationManager" enabled="true" />
  <component name="ProjectRootManager" version="2" languageLevel="JDK_11"
default="true" project-jdk-name="Android Studio default JDK" project-jdk-type="JavaSDK">
    <output url="file://$PROJECT_DIR$/build/classes" />
  </component>
  <component name="ProjectType">
    <option name="id" value="Android" />
  </component>
</project>

<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="VcsDirectoryMappings">
    <mapping directory="$PROJECT_DIR$" vcs="Git" />
  </component>
</project>

package com.example.expensestracker
import androidx.test.platform.app.InstrumentationRegistry
import androidx.test.ext.junit.runners.AndroidJUnit4
import org.junit.Test
import org.junit.runner.RunWith
import org.junit.Assert.*
/**
 * Instrumented test, which will execute on an Android device.
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
@RunWith(AndroidJUnit4::class)
class ExampleInstrumentedTest {
    @Test
    fun useAppContext() {
        // Context of the app under test.
        val appContext = InstrumentationRegistry.getInstrumentation().targetContext
        assertEquals("com.example.expensestracker", appContext.packageName)
    }
}

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

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

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

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

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

dependencies {
    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.3.1'
    implementation "androidx.compose.ui:ui:$compose_ui_version"
    implementation "androidx.compose.ui:ui-tooling-preview:$compose_ui_version"
    implementation 'androidx.compose.material:material:1.2.0'
    implementation 'androidx.room:room-common:2.5.1'
    implementation 'androidx.room:room-ktx:2.5.1'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
    androidTestImplementation "androidx.compose.ui:ui-test-junit4:$compose_ui_version"
    debugImplementation "androidx.compose.ui:ui-tooling:$compose_ui_version"
    debugImplementation "androidx.compose.ui:ui-test-manifest:$compose_ui_version"
}

# Add project specific ProGuard rules here.
# You can control the set of applied configuration files using the
# proguardFiles setting in build.gradle.
#
# For more details, see
#   http://developer.android.com/guide/developing/tools/proguard.html

# If your project uses WebView with JS, uncomment the following
# and specify the fully qualified class name to the JavaScript interface
# class:
#-keepclassmembers class fqcn.of.javascript.interface.for.webview {
#   public *;
#}

# Uncomment this to preserve the line number information for
# debugging stack traces.
#-keepattributes SourceFile,LineNumberTable

# If you keep the line number information, uncomment this to
# hide the original source file name.
#-renamesourcefileattribute SourceFile

#Mon Mar 27 10:00:30 IST 2023
distributionBase=GRADLE_USER_HOME
distributionUrl=https\://services.gradle.org/distributions/gradle-7.5-bin.zip
distributionPath=wrapper/dists
zipStorePath=wrapper/dists
zipStoreBase=GRADLE_USER_HOME

*.iml
.gradle
/local.properties
/.idea/caches
/.idea/libraries
/.idea/modules.xml
/.idea/workspace.xml
/.idea/navEditor.xml
/.idea/assetWizardSettings.xml
.DS_Store
/build
/captures
.externalNativeBuild
.cxx
local.properties

buildscript {
    ext {
        compose_ui_version = '1.2.0'
    }
}// Top-level build file where you can add configuration options common to all sub-projects/modules.
plugins {
    id 'com.android.application' version '7.4.1' apply false
    id 'com.android.library' version '7.4.1' apply false
    id 'org.jetbrains.kotlin.android' version '1.7.0' apply false
}

# Project-wide Gradle settings.
# IDE (e.g. Android Studio) users:
# Gradle settings configured through the IDE *will override*
# any settings specified in this file.
# For more details on how to configure your build environment visit
# http://www.gradle.org/docs/current/userguide/build_environment.html
# Specifies the JVM arguments used for the daemon process.
# The setting is particularly useful for tweaking memory settings.
org.gradle.jvmargs=-Xmx2048m -Dfile.encoding=UTF-8
# When configured, Gradle will run in incubating parallel mode.
# This option should only be used with decoupled projects. More details, visit
#
http://www.gradle.org/docs/current/userguide/multi_project_builds.html#sec:decoupled_projects
# org.gradle.parallel=true
# AndroidX package structure to make it clearer which packages are bundled with the
# Android operating system, and which are packaged with your app's APK
# https://developer.android.com/topic/libraries/support-library/androidx-rn
android.useAndroidX=true
# Kotlin code style for this project: "official" or "obsolete":
kotlin.code.style=official
```

```
# Enables namespacing of each library's R class so that its R class includes only the
# resources declared in the library itself and none from the library's dependencies,
# thereby reducing the size of the R class for that library
android.nonTransitiveRClass=true

#!/usr/bin/env sh

#
# Copyright 2015 the original author or authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#      https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

##############################################################################
##
##  Gradle start up script for UN*X
##
##############################################################################

# Attempt to set APP_HOME
# Resolve links: $0 may be a link
```

```
PRG="$0"
# Need this for relative symlinks.
while [ -h "$PRG" ] ; do
    ls=`ls -ld "$PRG"`
    link=`expr "$ls" : '.*-> \(.*\)$'`
    if expr "$link" : '/.*' > /dev/null; then
        PRG="$link"
    else
        PRG=`dirname "$PRG"`"/$link"
    fi
done
SAVED="`pwd`"
cd "`dirname \"$PRG\"`/" >/dev/null
APP_HOME="`pwd -P`"
cd "$SAVED" >/dev/null

APP_NAME="Gradle"
APP_BASE_NAME=`basename "$0"`

# Add default JVM options here. You can also use JAVA_OPTS and GRADLE_OPTS to pass JVM options to this script.
DEFAULT_JVM_OPTS='"-Xmx64m" "-Xms64m"'

# Use the maximum available, or set MAX_FD != -1 to use that value.
MAX_FD="maximum"

warn () {
    echo "$*"
}

die () {
    echo
    echo "$*"
    echo
    exit 1
}

# OS specific support (must be 'true' or 'false').
cygwin=false
msys=false
darwin=false
nonstop=false
case "`uname`" in
  CYGWIN* )
    cygwin=true
    ;;
  Darwin* )
    darwin=true
    ;;
  MINGW* )
    msys=true
    ;;
  NONSTOP* )
    nonstop=true
    ;;
esac

CLASSPATH=$APP_HOME/gradle/wrapper/gradle-wrapper.jar


# Determine the Java command to use to start the JVM.
if [ -n "$JAVA_HOME" ] ; then
```

```
    if [ -x "$JAVA_HOME/jre/sh/java" ] ; then
        # IBM's JDK on AIX uses strange locations for the executables
        JAVACMD="$JAVA_HOME/jre/sh/java"
    else
        JAVACMD="$JAVA_HOME/bin/java"
    fi
    if [ ! -x "$JAVACMD" ] ; then
        die "ERROR: JAVA_HOME is set to an invalid directory: $JAVA_HOME

Please set the JAVA_HOME variable in your environment to match the
location of your Java installation."
    fi
else
    JAVACMD="java"
    which java >/dev/null 2>&1 || die "ERROR: JAVA_HOME is not set and no 'java' command could be found in your PATH.

Please set the JAVA_HOME variable in your environment to match the
location of your Java installation."
fi

# Increase the maximum file descriptors if we can.
if [ "$cygwin" = "false" -a "$darwin" = "false" -a "$nonstop" = "false" ] ; then
    MAX_FD_LIMIT=`ulimit -H -n`
    if [ $? -eq 0 ] ; then
        if [ "$MAX_FD" = "maximum" -o "$MAX_FD" = "max" ] ; then
            MAX_FD="$MAX_FD_LIMIT"
        fi
        ulimit -n $MAX_FD
        if [ $? -ne 0 ] ; then
            warn "Could not set maximum file descriptor limit: $MAX_FD"
        fi
    else
        warn "Could not query maximum file descriptor limit: $MAX_FD_LIMIT"
    fi
fi

# For Darwin, add options to specify how the application appears in the dock
if $darwin; then
    GRADLE_OPTS="$GRADLE_OPTS                \"-Xdock:name=$APP_NAME\"                \"-
Xdock:icon=$APP_HOME/media/gradle.icns\""
fi

# For Cygwin or MSYS, switch paths to Windows format before running java
if [ "$cygwin" = "true" -o "$msys" = "true" ] ; then
    APP_HOME=`cygpath --path --mixed "$APP_HOME"`
    CLASSPATH=`cygpath --path --mixed "$CLASSPATH"`

    JAVACMD=`cygpath --unix "$JAVACMD"`

    # We build the pattern for arguments to be converted via cygpath
    ROOTDIRSRAW=`find -L / -maxdepth 1 -mindepth 1 -type d 2>/dev/null`
    SEP=""
    for dir in $ROOTDIRSRAW ; do
        ROOTDIRS="$ROOTDIRS$SEP$dir"
        SEP="|"
    done
    OURCYGPATTERN="(^($ROOTDIRS))"
    # Add a user-defined pattern to the cygpath arguments
    if [ "$GRADLE_CYGPATTERN" != "" ] ; then
        OURCYGPATTERN="$OURCYGPATTERN|($GRADLE_CYGPATTERN)"
    fi
    # Now convert the arguments - kludge to limit ourselves to /bin/sh
    i=0
    for arg in "$@" ; do
        CHECK=`echo "$arg"|egrep -c "$OURCYGPATTERN" -`
        CHECK2=`echo "$arg"|egrep -c "^-"`                          ### Determine if an option

        if [ $CHECK -ne 0 ] && [ $CHECK2 -eq 0 ] ; then                  ### Added a condition
            eval `echo args$i`=`cygpath --path --ignore --mixed "$arg"`
        else
            eval `echo args$i`="\"$arg\""
        fi
        i=`expr $i + 1`
    done
    case $i in
        0) set -- ;;
        1) set -- "$args0" ;;
        2) set -- "$args0" "$args1" ;;
        3) set -- "$args0" "$args1" "$args2" ;;
        4) set -- "$args0" "$args1" "$args2" "$args3" ;;
        5) set -- "$args0" "$args1" "$args2" "$args3" "$args4" ;;
        6) set -- "$args0" "$args1" "$args2" "$args3" "$args4" "$args5" ;;
        7) set -- "$args0" "$args1" "$args2" "$args3" "$args4" "$args5" "$args6" ;;
        8) set -- "$args0" "$args1" "$args2" "$args3" "$args4" "$args5" "$args6" "$args7" ;;
        9) set -- "$args0" "$args1" "$args2" "$args3" "$args4" "$args5" "$args6" "$args7" "$args8" ;;
    esac
fi

# Escape application args
save () {
    for i do printf %s\\n "$i" | sed "s/'/'\\\\''/g;1s/^/'/;\$s/\$/' \\\\/" ; done
    echo " "
```

```
}
APP_ARGS=`save "$@"`

# Collect all arguments for the java command, following the shell quoting and substitution rules
eval set -- $DEFAULT_JVM_OPTS $JAVA_OPTS $GRADLE_OPTS "\"-Dorg.gradle.appname=$APP_BASE_NAME\"" -classpath "\"$CLASSPATH\"" org.gradle.wrapper.GradleWrapperMain "$APP_ARGS"

exec "$JAVACMD" "$@"

pluginManagement {
   repositories {
      google()
      mavenCentral()
      gradlePluginPortal()
   }
}
dependencyResolutionManagement {
   repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
   repositories {
      google()
      mavenCentral()
   }
}
rootProject.name = "Expenses Tracker"
include ':app'

package com.example.expensestracker


import org.junit.Test
import org.junit.Assert.*
/**
 * Example local unit test, which will execute on the development machine (host).
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
class ExampleUnitTest {
    @Test
    fun addition_isCorrect() {
        assertEquals(4, 2 + 2)
    }
}
?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.ExpensesTracker"
        tools:targetApi="31">
        <activity
            android:name=".RegisterActivity"
            android:exported="false"
            android:label="@string/title_activity_register"
            android:theme="@style/Theme.ExpensesTracker" />
        <activity
            android:name=".MainActivity"
            android:exported="false"
            android:label="MainActivity"
            android:theme="@style/Theme.ExpensesTracker" />
        <activity
            android:name=".ViewRecordsActivity"
            android:exported="false"
            android:label="@string/title_activity_view_records"
            android:theme="@style/Theme.ExpensesTracker" />
        <activity
            android:name=".SetLimitActivity"
            android:exported="false"
            android:label="@string/title_activity_set_limit"
            android:theme="@style/Theme.ExpensesTracker" />
        <activity
            android:name=".AddExpensesActivity"
            android:exported="false"
            android:label="@string/title_activity_add_expenses"
            android:theme="@style/Theme.ExpensesTracker" />
        <activity
            android:name=".LoginActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.ExpensesTracker">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>  </application>
</manifest>
<vector xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:aapt="http://schemas.android.com/aapt"
    android:width="108dp"
    android:height="108dp"
    android:viewportWidth="108"
    android:viewportHeight="108">
    <path android:pathData="M31,63.928c0,0 6.4,-11 12.1,-13.1c7.2,-2.6 26,-1.4 26,-
1.4l38.1,38.1L107,108.928l-32,-1L31,63.928z">
        <aapt:attr name="android:fillColor">
            <gradient
                android:endX="85.84757"
                android:endY="92.4963"
                android:startX="42.9492"
                android:startY="49.59793"
                android:type="linear">
                <item
                    android:color="#44000000"
                    android:offset="0.0" />
                <item
                    android:color="#00000000"
                    android:offset="1.0" />
            </gradient>
        </aapt:attr>
    </path>
    <path
        android:fillColor="#FFFFFF"
        android:fillType="nonZero"
        android:pathData="M65.3,45.828l3.8,-6.6c0.2,-0.4 0.1,-0.9 -0.3,-1.1c-0.4,-0.2 -0.9,-0.1 -
1.1,0.3l-3.9,6.7c-6.3,-2.8 -13.4,-2.8 -19.7,0l-3.9,-6.7c-0.2,-0.4 -0.7,-0.5 -1.1,-0.3C38.8,38.328
38.7,38.828 38.9,39.228l3.8,6.6C36.2,49.428 31.7,56.028 31,63.928h46C76.3,56.028
71.8,49.428 65.3,45.828zM43.4,57.328c-0.8,0 -1.5,-0.5 -1.8,-1.2c-0.3,-0.7 -0.1,-1.5 0.4,-2.1c0.5,-
0.5 1.4,-0.7 2.1,-0.4c0.7,0.3 1.2,1 1.2,1.8C45.3,56.528 44.5,57.328
```

```
43.4,57.328L43.4,57.328zM64.6,57.328c-0.8,0 -1.5,-0.5 -1.8,-1.2s-0.1,-1.5 0.4,-2.1c0.5,-0.5 1.4,-
0.7 2.1,-0.4c0.7,0.3 1.2,1 1.2,1.8C66.5,56.528 65.6,57.328 64.6,57.328L64.6,57.328z"
        android:strokeWidth="1"
        android:strokeColor="#00000000" />
</vector>

<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/android">
    <background android:drawable="@drawable/ic_launcher_background" />
    <foreground android:drawable="@drawable/ic_launcher_foreground" />
</adaptive-icon>

<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/android">
    <background android:drawable="@drawable/ic_launcher_background" />
    <foreground android:drawable="@drawable/ic_launcher_foreground" />
</adaptive-icon>

<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/android">
    <background android:drawable="@drawable/ic_launcher_background" />
    <foreground android:drawable="@drawable/ic_launcher_foreground" />
    <monochrome android:drawable="@drawable/ic_launcher_foreground" />
</adaptive-icon>

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
</resources>

<resources>
    <string name="app_name">Expenses Tracker</string>
    <string name="title_activity_add_expenses">AddExpensesActivity</string>
    <string name="title_activity_set_limit">SetLimitActivity</string>
    <string name="title_activity_view_records">ViewRecordsActivity</string>
    <string name="title_activity_login">LoginActivity</string>
    <string name="title_activity_register">RegisterActivity</string>
</resources>

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <stylename="Theme.ExpensesTracker"
parent="android:Theme.Material.Light.NoActionBar">
        <item name="android:statusBarColor">@color/purple_700</item>
    </style>
</resources>

<?xml version="1.0" encoding="utf-8"?><!--
   Sample backup rules file; uncomment and customize as necessary.
   See https://developer.android.com/guide/topics/data/autobackup
   for details.
   Note: This file is ignored for devices older that API 31
   See https://developer.android.com/about/versions/12/backup-restore
-->
<full-backup-content>
    <!--
   <include domain="sharedpref" path="."/>
   <exclude domain="sharedpref" path="device.xml"/>
-->
</full-backup-content>

<?xml version="1.0" encoding="utf-8"?><!--
   Sample data extraction rules file; uncomment and customize as necessary.
   See https://developer.android.com/about/versions/12/backup-restore#xml-changes
   for details.
-->
<data-extraction-rules>
    <cloud-backup>
        <!-- TODO: Use <include> and <exclude> to control what is backed up.
        <include .../>
        <exclude .../>
        -->
    </cloud-backup>
    <!--
    <device-transfer>
        <include .../>
        <exclude .../>
    </device-transfer>
    -->
</data-extraction-rules>

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")
        }
    }
}

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?,
)

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)
   susped fun insertExpense(items: Expense)
   @Update
   suspend fun updateExpense(items: Expense)
   @Delete
   suspend fun deleteExpense(items: Expense)
}
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
        prvate 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
            }
        }
    }
}
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
  }
}

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.expensestracker

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


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

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

        private const val TABLE_NAME = "items_table"
        private const val COLUMN_ID = "id"
        private const val COLUMN_ITEM_NAME = "item_name"
        private const val COLUMN_QUANTITY = "quantity"
        private const val COLUMN_COST = "cost"
    }

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

        db?.execSQL(createTable)
    }

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

    fun insertItems(items: Items) {
        val db = writableDatabase
        val values = ContentValues()
        values.put(COLUMN_ITEM_NAME, items.itemName)
        values.put(COLUMN_QUANTITY, items.quantity)
        values.put(COLUMN_COST, items.cost)
        db.insert(TABLE_NAME, null, values)
        db.close()
    }
```

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

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

```
package com.example.expensestracker

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

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

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

        )

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

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

```
) {

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

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

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

    )

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

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

            } else {
                error = "Please fill all fields"
            }
        },
        modifier = Modifier.padding(top = 16.dp)
    ) {
        Text(text = "Login")
```

```
        }
        Row {
            TextButton(onClick = {context.startActivity(
                Intent(
                    context,
                    RegisterActivity::class.java
                )
            )}
            )
            { Text(color = Color.White,text = "Sign up") }
            TextButton(onClick = {
            })

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