BCA 0416

Money Management

A PROJECT REPORT SUBMITTED TO GOA UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENT

FOR THE DEGREE OF BCA

BY

Rekha Paranjape

Anuja Jawadekar

Vallabh Naik

Mayur Pilyenkar

[illegible]	[illegible]	[illegible] 15/03/17
Sameer Patil	R. JebaRaj	Dr. Manasvi M. Kamat
Internal Guide	Project Coordinator	Principal

Goa Multi - Faculty College

2016-2017

DECLARATION BY CANDIDATES

We declare that this project report has been prepared by us and to the best of our knowledge, it has not previously formed the basis for the award of any diploma or degree by this or any other University.

Roll No	Name	Signature
4328	Rekha Paranjape	[illegible]
4311	Anuja Jawadekar	[illegible]
4327	Vallabh Naik	[illegible]
4329	Mayur Pilyankar	[illegible]

Shree Sateri Pissani Education Society's

Shri. Gopal Gaonkar Memorial

Goa Multi - Faculty College

Dharbandora - Goa

Affiliated to Goa University

CERTIFICATE

This is to certify that a project on Money Management has been successfully completed by 1)Rekha Paranjape 2)Anuja Jawadekar 3)Vallabh Naik 4)Mayur Pilyenkar studying in T.Y.B.C.A during the academic year 2016-2017. The project has been carried out under the supervision of the internal guide.

[illegible] | [illegible] (MELISA RODRIGUES) | [illegible] 15/03/17

Sameer Patil | | Dr. Manasvi M. Kamat

Internal Guide | External Examiner | Principal

Principal
Shri. Gopal Gaonkar Memorial
GOA MULTI FACULTY COLLEGE

[illegible]

Project Coordinator

Place: Dharbandora

Date: 10/03/2017

SSPES's Dharbandora Goa Goa Multi-Faculty College

IV.ACKNOWLEDGEMENT

We hereby take the opportunity to express our deep sense of gratitude to all those who gave their wholehearted co-operation in the completion of our project.

The task we have undertaken wouldn't have evolved successfully, without the help of some eminent personalities who have indeed left a deep impact on our mind.

We are grateful to our respected principal Dr.**Manasvi Kamat** whose sense of motivation and discipline inspired us a lot.

We express our thanks to our H.O.D Mr. **R. Jebaraj** for providing us all the facilities during our project work.

Firstly, we would like to extend our greatest gratitude to our guide Mr. **SAMEER PATIL** who has been always a source of enlight for us in assisting and supporting us in our work.

Contents

1.Introduction

In today`s technical world most of the things are becoming faster and easier and things are working on a tip of a finger.

Having an application on a cell phone which will make ones day to day life easy can be can be done with less effort and can be made more easy. This is what gave us thought of developing an application named"Money Manage" .This is the name of our project "Money management Application.

As its name suggests, this application is for cash management.

This application allows user to maintain their day to day earnings and expenditure and can help to plan their budget. They can find out their balance at the end of the month, week or a day. This application is going to be very helpful in maintaining cash.

User can select the date and can enter his expenses or incomes in this application.

As it provides fields to enter the description about the income or expense, it is useful for the user.

Each person's expenses and income field can be different, so this application is useful for each person.

One person's expense category can be a income for the other person. So, he can add his category name and select whether it is income or expense and enter the amount.

2.Features of proposed system

This application gives user a relief from maintaining a diary for recording his incomes and expenses to keep track of his money.

This application helps user to keep track of his income and expense by just adding his expense or income amount in this app.

User will have to add category and amount he spent on it or earned.

User can see his income and expenses and can know his balance.

User can track his income and expenses and get overall view of income and expense.

3.Entity Relationship Diagram

An entity-relationship diagram (ERD) is a data modeling technique that graphically illustrates an information system's entities and the relationships between those entities. An ERD is a conceptual and representational model of data used to represent the entity framework infrastructure.

The elements of an ERD are:

- Entities
- Relationships
- Attribute

Steps involved in creating an ERD include:

1. Identifying and defining the entities
2. Determining all interactions between the entities
3. Analyzing the nature of interactions/determining the cardinality of the relationships
4. Creating the ERD

An entity-relationship diagram (ERD) is crucial to creating a good database design. It is used as a high-level logical data model, which is useful in developing a conceptual design for databases.

An entity is a real-world item or concept that exists on its own. Entities are equivalent to database tables in a relational database, with each row of the table representing an instance of that entity.

An attribute of an entity is a particular property that describes the entity. A relationship is the association that describes the interaction between entities. Cardinality, in the context of ERD, is the number of instances of one entity that can, or must, be associated with each instance of another entity. In general, there may be one-to-one, one-to-many, or many-to-many relationships.

4.Activity Diagram

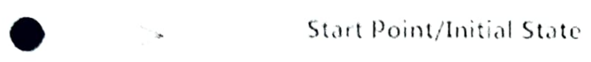
An activity diagram visually presents a series of actions or flow of control in a system similar to a flowchart or a data flow diagram. Activity diagrams are often used in business process modeling. They can also describe the steps in a use case diagram. Activities modeled can be sequential and concurrent. In both cases an activity diagram will have a beginning and an end.

An Activity Diagram is a behavioral diagram that shows the flow or sequence of activities through a system. The terms activity diagram and process flow are often used interchangeably. However, the term activity diagram is typically more restrictive as it refers to one of thirteen standard Unified Model Language (UML) diagrams. Activity Diagrams are one of the most commonly used diagrams since its notation and origin are based on the widely known flowchart notation.

Basic Activity Diagram Notations and Symbols

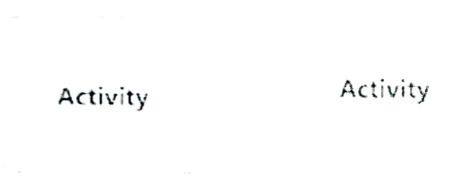
Initial State or Start Point

A small filled circle followed by an arrow represents the initial action state or the start point for any activity diagram. For activity diagram using swimlanes, make sure the start point is placed in the top left corner of the first column.



Activity or Action State

An action state represents the non-interruptible action of objects. You can draw an action state in SmartDraw using a rectangle with rounded corners.



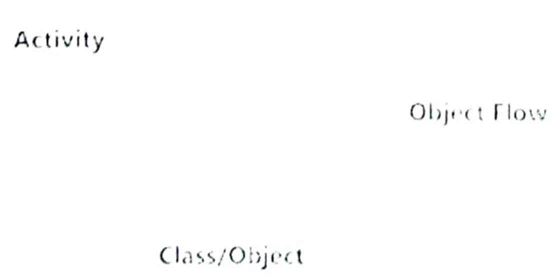
Action Flow

Action flows, also called edges and paths, illustrate the transitions from one action state to another. They are usually drawn with an arrowed line.

Action Flow

Object Flow

Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.



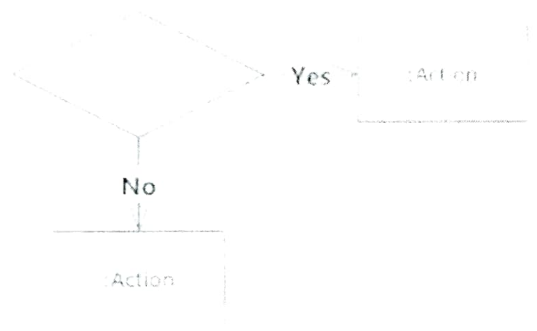
Decisions and Branching

A diamond represents a decision with alternate paths. When an activity requires a decision prior to moving on to the next activity, add a diamond between the two activities. The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else."

Decision Symbol

Guards

In UML, guards are a statement written next to a decision diamond that must be true before moving next to the next activity. These are not essential, but are useful when a specific answer, such as "Yes, three labels are printed," is needed before moving forward.



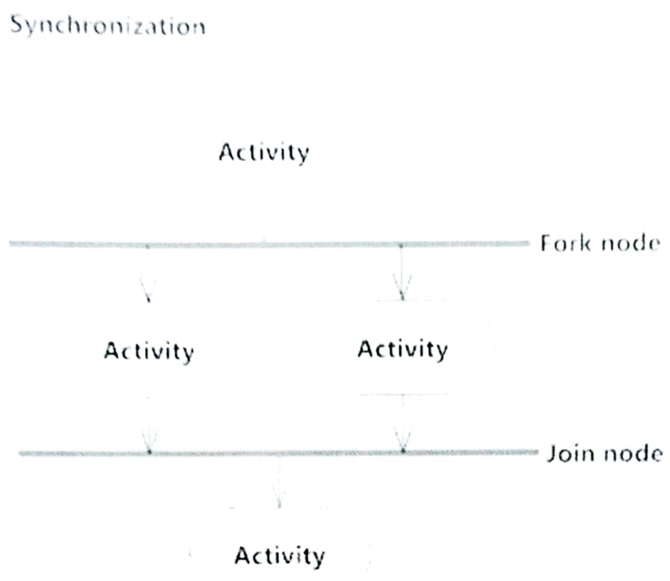
Guard Symbols

Synchronization

A fork node is used to split a single incoming flow into multiple concurrent flows. It is represented as a straight, slightly thicker line in an activity diagram.

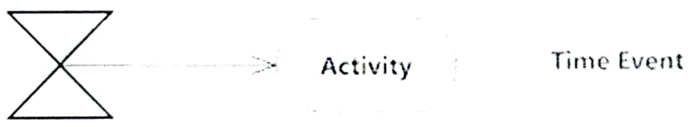
A join node joins multiple concurrent flows back into a single outgoing flow.

A fork and join mode used together are often referred to as synchronization.



Time Event

This refers to an event that stops the flow for a time; an hourglass depicts it.

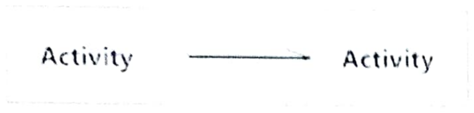


Merge Event

A merge event brings together multiple flows that are not concurrent.

Sent and Received Signals

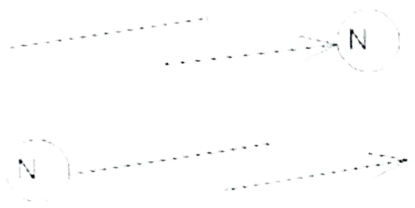
Signals represent how activities can be modified from outside the system. They usually appear in pairs of sent and received signals, because the state can't change until a response is received, much like synchronous messages in a sequence diagram. For example, an authorization of payment is needed before an order can be completed.



Signal sent and received

Interrupting Edge

An event, such as a cancellation, that interrupts the flow denoted with a lightning bolt.

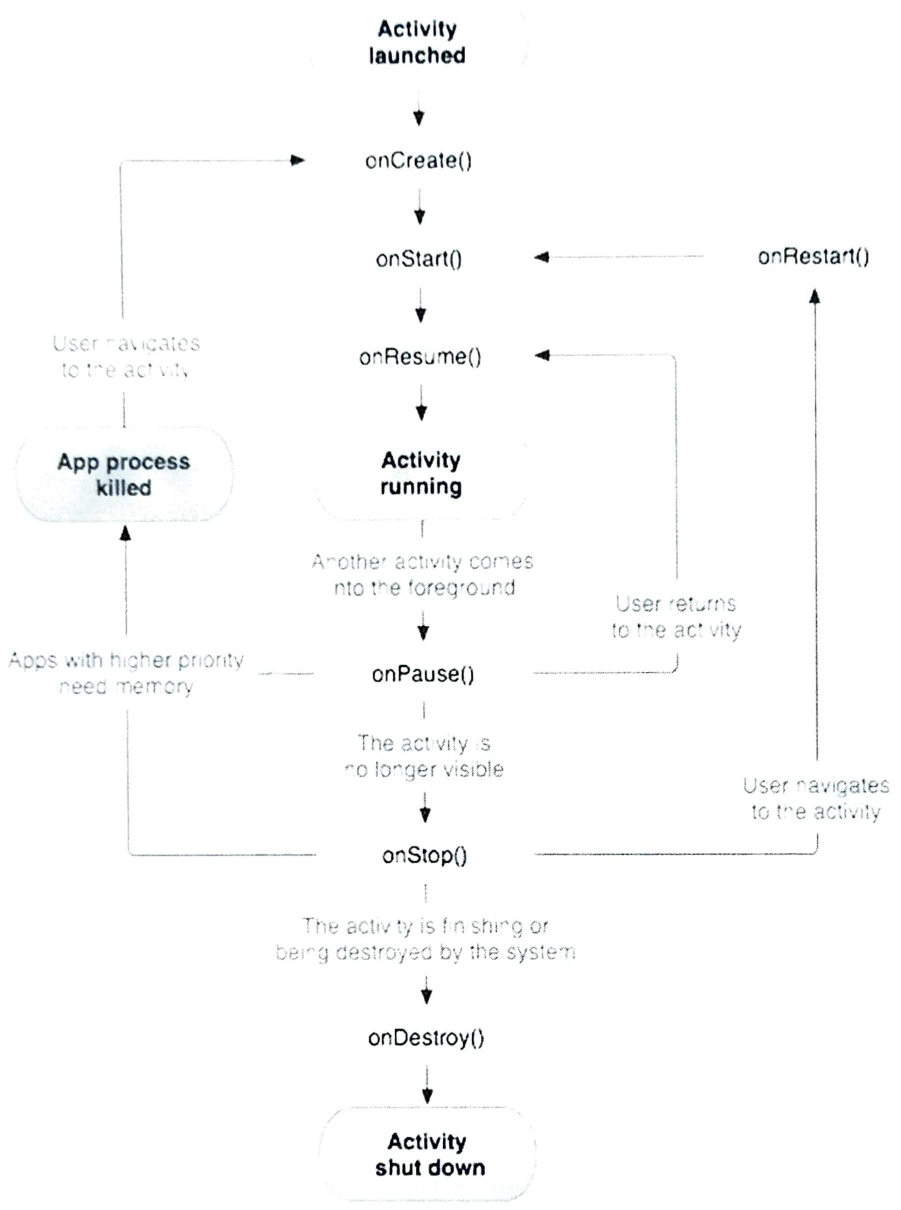


Interrupting Edge Symbols

Purpose of Activity Diagram

Purpose: The basic **purposes of activity diagrams** are similar to other four **diagrams**. It captures the dynamic behaviour of the system. Other four **diagrams** are used to show the message flow from one object to another but **activity diagram** is used to show message flow from one **activity** to another.

Activity Life Cycle



Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Oncreate()

Called when the activity is first created. This is where you should do all of your normal static set up: create views, bind data to lists, etc. This method also provides you with a Bundle containing the activity's previously frozen state, if there was one. Always followed by onStart().

onRestart():

Called after your activity has been stopped, prior to it being started again. Always followed by onStart()

onStart():

Called when the activity is becoming visible to the user. Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden.

onResume():

Called when the activity will start interacting with the user. At this point your activity is at the top of the activity stack, with user input going to it. Always followed by onPause().

onPause ():

Called as part of the activity lifecycle when an activity is going into the background, but has not (yet) been killed. The counterpart to onResume(). When activity B is launched in front of activity A, this callback will be invoked on A. B will not be created until A's onPause() returns, so be sure to not do anything lengthy here.

onStop():

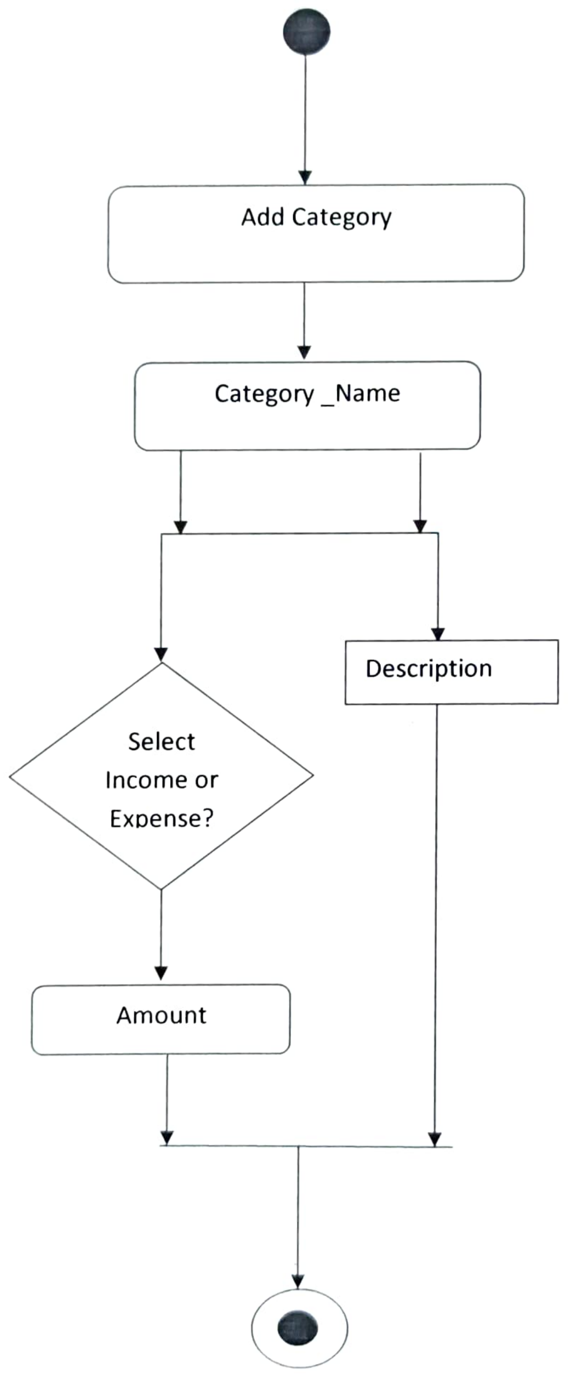
Called when you are no longer visible to the user. You will next receive either onRestart(), onDestroy(), or nothing, depending on later user activity.

Note that this method may never be called, in low memory situations where the system does not have enough memory to keep your activity's process running after its onPause() method is called.

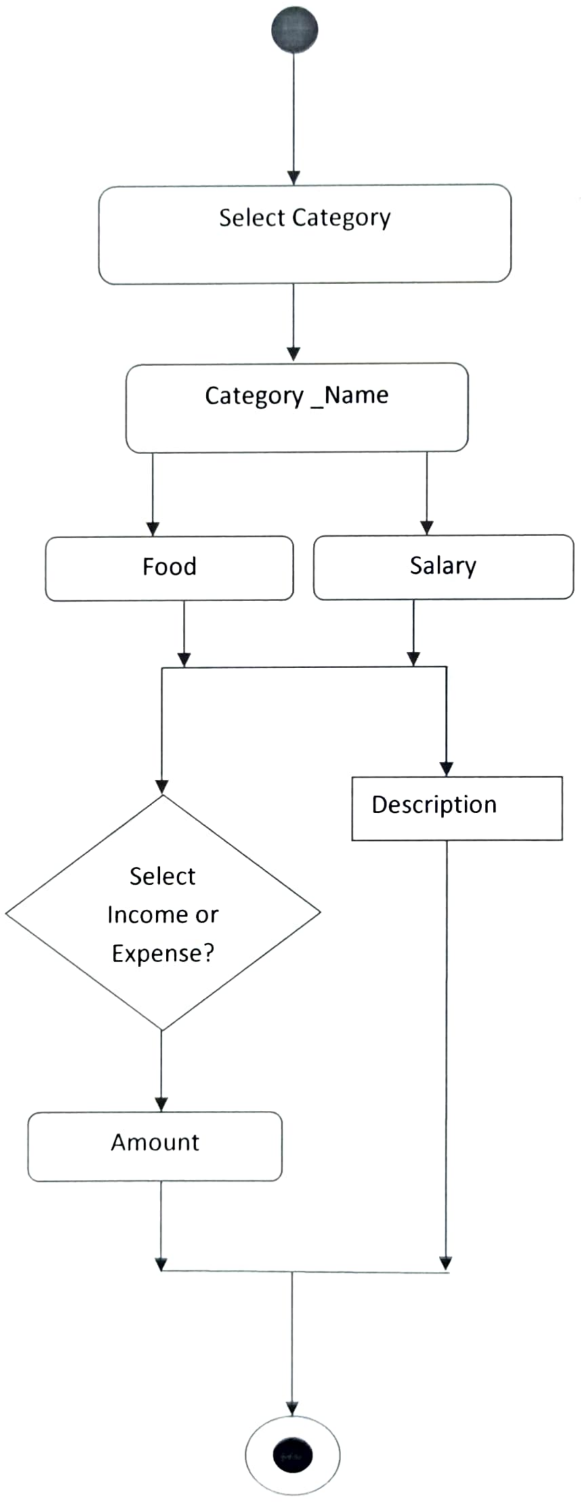
onDestroy():

The final call you receive before your activity is destroyed. This can happen either because the activity is finishing (someone called finish() on it, or because the system is temporarily destroying this instance of the activity to save space. You can distinguish between these two scenarios with the isFinishing() method.

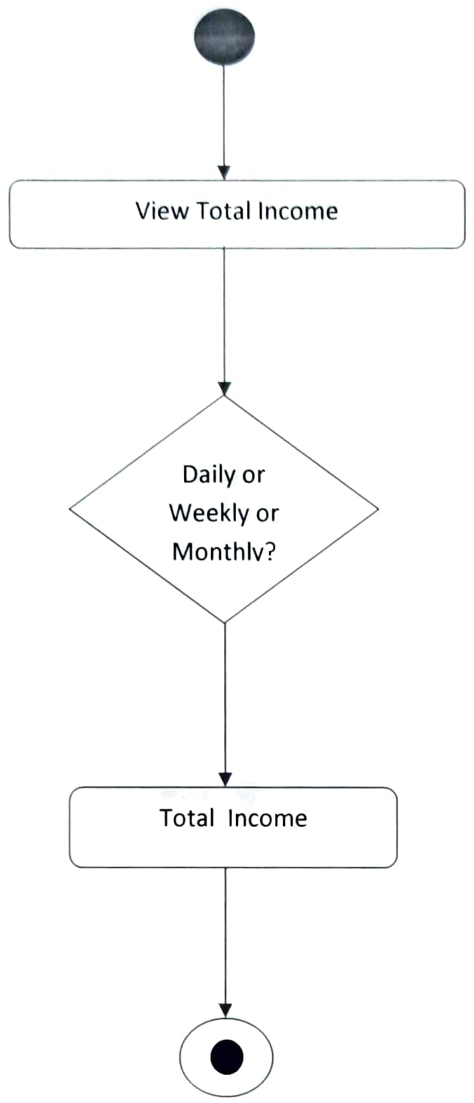
Add Category Activity Diagram



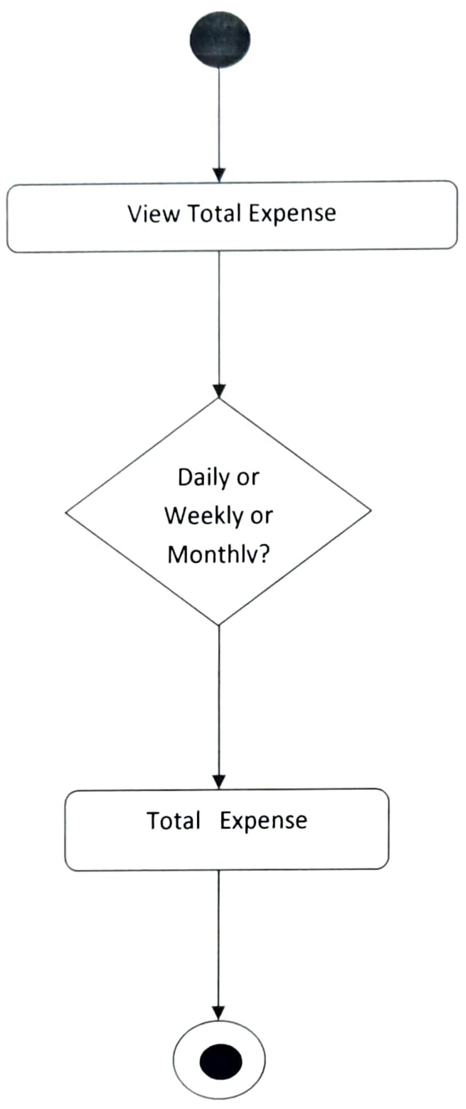
Select Category Activity Diagram



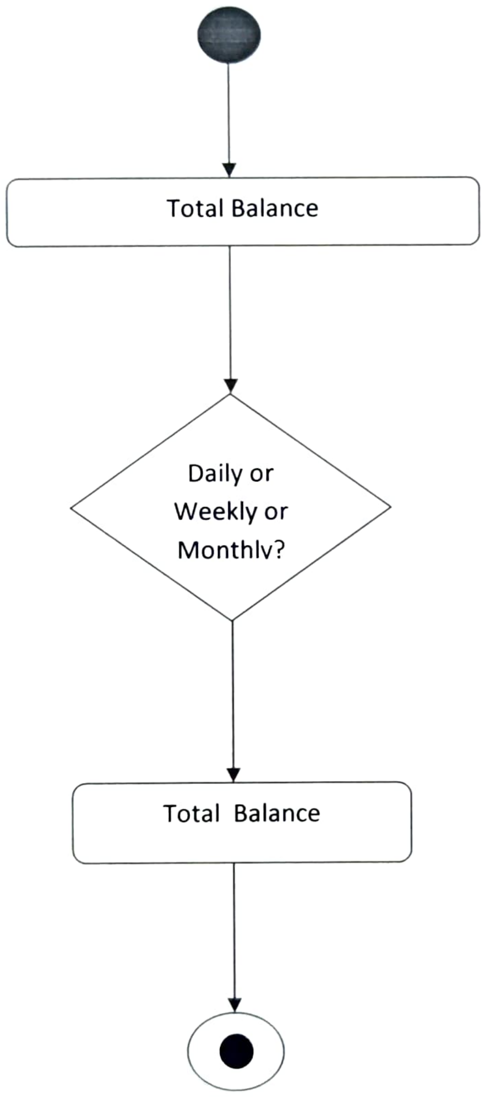
View Total Income Activity Diagram



View Total Expense Activity Diagram



Total Balance Activity Diagram



5.USE CASE DIAGRAM

A use case is a software and system engineering term that describes how a user uses a system to accomplish a particular goal. A use case acts as a software modeling technique that defines the features to be implemented and the resolution of any errors that may be encountered.

Use cases define interactions between external actors and the system to attain particular goals. There are three basic elements that make up a use case:

- Actors: Actors are the type of users that interact with the system.
- System: Use cases capture functional requirements that specify the intended behavior of the system.
- Goals: Use cases are typically initiated by a user to fulfill goals describing the activities and variants involved in attaining the goal.

Use cases are modeled using unified modeling language and are represented by ovals containing the names of the use case. Actors are represented using lines with the name of the actor written below the line. To represent an actor's participation in a system, a line is drawn between the actor and the use case. Boxes around the use case represent the system boundary.

Characteristics associated with use cases are:

- Organizing functional requirements
- Modeling the goals of system user interactions
- Recording scenarios from trigger events to ultimate goals
- Describing the basic course of actions and exceptional flow of events
- Permitting a user to access the functionality of another event

The steps in designing use cases are:

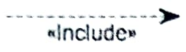
- Identify the users of the system
- For each category of users, create a user profile. This includes all roles played by the users relevant to the system.
- Identify significant goals associated with each role to support the system. The system's value proposition identifies the significant role.
- Create use cases for every goal associated with a use case template and maintain the same abstraction level throughout the use case. Higher level use case steps are treated as goals for the lower level.
- Structure the use cases
- Review and validate the users

Purpose of Use Case Diagram

The main **purpose** of a **use case diagram** is to show who interacts with your system, and the main goals they achieve with it. Create Actors to represent classes of people, organizations, other systems, software or devices that interact with your system or subsystem.

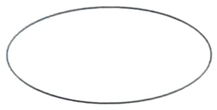
Elements of Use Case

An include relationship is a relationship between two use cases:



It indicates that the use case to which the arrow points is included in the use case on the other side of the arrow. This makes it possible to reuse a use case in another use case.

Use Case



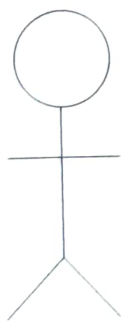
Use cases describe the interactions that take place between actors and IT systems during the execution of business processes:

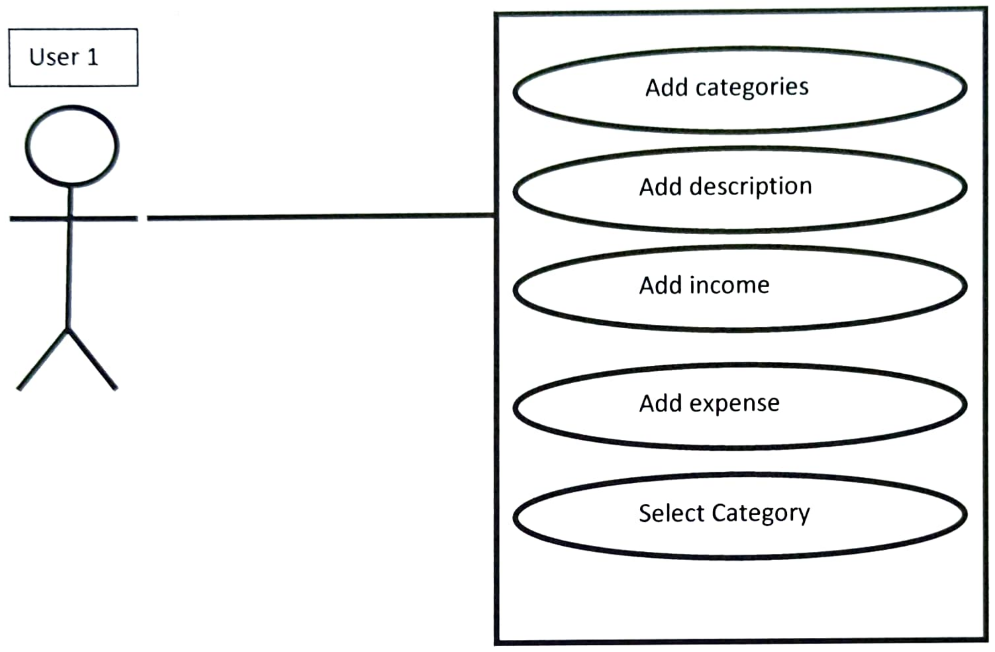
Association

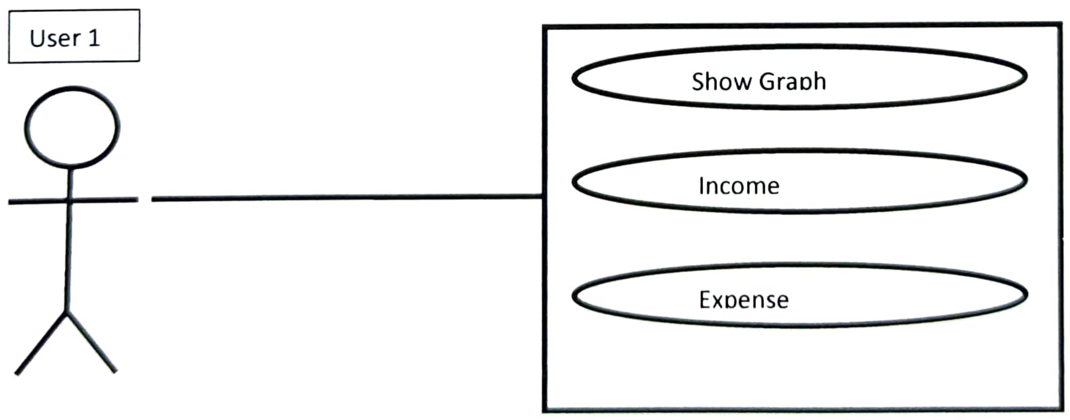
An association is a connection between an actor and a use case. An association indicates that an actor can carry out a use case.

Actor

Actors represent roles that users take on when they use the IT system




User 1
Add categories
Add description
Add income
Add expense
Select Category


User 1
Show Graph
Income
Expense

6.Back-end Tools

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endianarchitectures. These features make SQLite a popular choice as an Application File Format. Think of SQLite not as a replacement for Oracle but as a replacement for fopen()

SQLite is a compact library. With all features enabled, the library size can be less than 500KiB, depending on the target platform and compiler optimization settings. (64-bit code is larger. And some compiler optimizations such as aggressive function inlining and loop unrolling can cause the object code to be much larger.) If optional features are omitted, the size of the SQLite library can be reduced below 300KiB. SQLite can also be made to run in minimal stack space (4KiB) and very little heap (100KiB), making SQLite a popular database engine choice on memory constrained gadgets such as cellphones, PDAs, and MP3 players. There is a tradeoff between memory usage and speed. SQLite generally runs faster the more memory you give it. Nevertheless, performance is usually quite good even in low-memory environments. SQLite is very carefully tested prior to every release and has a reputation for being very reliable. Most of the SQLite source code is devoted purely to testing and verification. An automated test suite runs millions and millions of test cases involving hundreds of millions of individual SQL statements and achieves 100% branch test coverage. SQLite responds gracefully to memory allocation failures and disk I/O errors. Transactions are ACID even if interrupted by system crashes or power failures. All of this is verified by the automated tests using special test harnesses which simulate system failures. Of course, even with all this testing, there are still bugs. But unlike some similar projects (especially commercial competitors) SQLite is open and honest about all bugs and provides bugs lists and minute-by-minute chronologies of code changes.

The SQLite code base is supported by an international team of developers who work on SQLite full-time. The developers continue to expand the capabilities of SQLite and enhance its reliability and performance while maintaining backwards compatibility with the published interface spec, SQL syntax, and database file format. The source code is absolutely free to anybody who wants it, but professional support is also available.

The SQLite project was started on 2000-05-09. The future is always hard to predict, but the intent of the developers is to support SQLite through the year 2050. Design decisions are made with that objective in mind.

We the developers hope that you find SQLite useful and we entreat you to use it well: to make good and beautiful products that are fast, reliable, and simple to use. Seek forgiveness for yourself as you forgive others. And just as you have received SQLite for free, so also freely give, paying the debt forward.

7.Front-end Tools

Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA . On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Instant Run to push changes to your running app without building a new APK
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

This page provides an introduction to basic Android Studio features. For a summary of the latest changes, see Android Studio Release Notes.

Project Structure

Each project in Android Studio contains one or more modules with source code files and resource files. Types of modules include:

- Android app modules
- Library modules
- Google App Engine modules

By default, Android Studio displays your project files in the Android project view, as shown in figure 1. This view is organized by modules to provide quick access to your project's key source files.

All the build files are visible at the top level under **Gradle Scripts** and each app module contains the following folders:

- **manifests**: Contains the AndroidManifest.xml file.
- **java**: Contains the Java source code files, including JUnit test code.
- **res**: Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select **Project** from the **Project** dropdown (in figure 1, it's showing as**Android**).

You can also customize the view of the project files to focus on specific aspects of your app development. For example, selecting the **Problems** view of your project displays links to the source files containing any recognized coding and syntax errors, such as a missing XML element closing tag in a layout file.

The User Interface

1. The **toolbar** lets you carry out a wide range of actions, including running your app and launching Android tools.
2. The **navigation bar** helps you navigate through your project and open files for editing. It provides a more compact view of the structure visible in the **Project** window.
3. The **editor window** is where you create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.
4. The **tool window bar** runs around the outside of the IDE window and contains the buttons that allow you to expand or collapse individual tool windows.
5. The **tool windows** give you access to specific tasks like project management, search, version control, and more. You can expand them and collapse them.
6. The **status bar** displays the status of your project and the IDE itself, as well as any warnings or messages.

You can organize the main window to give yourself more screen space by hiding or moving toolbars and tool windows. You can also use keyboard shortcuts to access most IDE features.

At any time, you can search across your source code, databases, actions, elements of the user interface, and so on, by double-pressing the Shift key, or clicking the magnifying glass in the upper right-hand corner of the Android Studio window. This can be very useful if, for example, you are trying to locate a particular IDE action that you have forgotten how to trigger.

Tool Windows
Instead of using preset perspectives, Android Studio follows your context and automatically brings up relevant tool windows as you work. By default, the most commonly used tool windows are pinned to the tool window bar at the edges of the application window.

- To expand or collapse a tool window, click the tool's name in the tool window bar. You can also drag, pin, unpin, attach, and detach tool windows.
- To return to the current default tool window layout, click **Window > Restore Default Layout** or customize your default layout by clicking **Window > Store Current Layout as Default**.

- To show or hide the entire tool window bar, click the window icon in the bottom left-hand corner of the Android Studio window.
- To locate a specific tool window, hover over the window icon and select the tool window from the menu.

8.Middleware And Auxillary Tools

XML

1) XML stands for eXtensible Markup Language
2) XML is a markup language much like HTML
3) XML was designed to store and transport data
4) XML was designed to be self-descriptive

XML (or markup languages derived from it) is used quite a lot in data transfer because it's an intermediate, platform independent format. **XML** Schema can be used to enforce a particular structure and form in an **XML** document, and thus can be used to create custom markup languages.

According to new design pattern in android studio activity_main.xml will determine how the global UI of the Activity should be. And on the other hand content_main.xml will determine the contents in the activity_main.xml.

That is content_main.xml will contain the textview, edittext, button etc component. And it will be included by the activity_main.xml.

So we can think of content_main.xml just like partial in HTML. activity_main.xml will contain your activity global design, and content_main.xml will contain the contents.

9.Validation Test Report

In computer science, data **validation** is the process of ensuring that a **program** operates on clean, correct and useful data. It uses routines, often called "**validation** rules" "**validation** constraints" or "check routines", that check for correctness, meaningfulness, and security of data that are input to the system.

Types of validation

- Data type validation;
- Range and constraint validation;
- Code and Cross-reference validation; and
- Structured validation

Data type validation

Data type validation is customarily carried out on one or more simple data fields.

The simplest kind of data type validation verifies that the individual characters provided through user input are consistent with the expected characters of one or more known primitive data types; as defined in a programming language or data storage and retrieval mechanism.For example, many database systems allow the specification of the following primitive data types: 1) integer; 2) float (decimal); or 3) string.

Similarly, telephone numbers are routinely expected to include the digits and possibly the characters +, -, (, and) (plus, minus, and parentheses). A more sophisticated data validation routine would check to see the user had entered a valid country code, i.e., that the number of digits entered matched the convention for the country or area specified.

A validation process involves two distinct steps: (a) Validation Check and (b) Post-Check action. The check step uses one or more computational rules (see section below) to determine if the data is valid. The Post-validation action sends feedback to help enforce validation.

Simple range and constraint validation

Simple range and constraint validation may examine user input for consistency with a minimum/maximum range, or consistency with a test for evaluating a sequence of characters, such as one or more tests against regular expressions.

Code and cross-reference validation

Code and cross-reference validation includes tests for data type validation, combined with one or more operations to verify that the user-supplied data is consistent with one or more external rules, requirements, or validity constraints relevant to a particular organization, context or set of underlying assumptions. These additional validity constraints may involve cross-referencing supplied data with a known look-up table or directory information service such as LDAP.

For example, a experienced user may enter a well-formed string that matches the specification for a valid e-mail address, as defined in RFC 5322,but that well-formed string might not actually correspond to a resolvable domain connected to an active e-mail account.

Structured validation

Structured validation allows for the combination of any of various basic data type validation steps, along with more complex processing. Such complex processing may include the testing of conditional constraints for an entire complex data object or set of process operations within a system.

Validation methods

Allowed character checks

Checks to ascertain that only expected characters are present in a field. For example a numeric field may only allow the digits 0–9, the decimal point and perhaps a minus sign or commas. A text field such as a personal name might disallow characters such as < and >, as they could be evidence of a markup-based security attack. An e-mail address might require at least one @ sign and various other structural details. Regular expressions are effective ways of implementing such checks. (See also data type checks below)

Batch totals

Checks for missing records. Numerical fields may be added together for all records in a batch. The batch total is entered and the computer checks that the total is correct, e.g., add the 'Total Cost' field of a number of transactions together.

Cardinality check

Checks that record has a valid number of related records. For example if Contact record classified as a Customer it must have at least one associated Order (Cardinality > 0). If order does not exist for a "customer" record then it must be either changed to "seed" or the order must be created. This type of rule can be complicated by additional conditions. For example if contact record in Payroll database is marked as "former employee", then this record must not have any associated salary payments after the date on which employee left organization (Cardinality = 0).

Check digits,

Used for numerical data. An extra digit is added to a number which is calculated from the digits. The computer checks this calculation when data are entered. For example the last digit of an ISBN for a book is a check digit calculated modulus 10.[3]

Consistency checks

Checks fields to ensure data in these fields corresponds, e.g., If Title = "Mr.", then Gender = "M".

Control totals

This is a total done on one or more numeric fields which appears in every record. This is a meaningful total, e.g., add the total payment for a number of Customers.

Cross-system consistency checks

Compares data in different systems to ensure it is consistent, e.g., The address for the customer with the same id is the same in both systems. The data may be represented differently in different systems and may need to be transformed to a common format to be compared, e.g., one system may store customer name in a single Name field as 'Doe, John Q', while another in three different fields: First_Name (John), Last_Name (Doe) and Middle_Name (Quality); to compare the two, the validation engine would have to transform data from the second system to match the data from the first, for example, using SQL: Last_Name || ', ' || First_Name || substr(Middle_Name, 1, 1) would convert the data from the second system to look like the data from the first 'Doe, John Q'

Data type checks

Checks the data type of the input and give an error message if the input data does not match with the chosen data type, e.g., In an input box accepting numeric data, if the letter 'O' was typed instead of the number zero, an error message would appear.

File existence check

Checks that a file with a specified name exists. This check is essential for programs that use file handling.

Format or picture check

Checks that the data is in a specified format (template), e.g., dates have to be in the format DD/MM/YYYY. Regular expressions should be considered for this type of validation.

Hash totals

This is just a batch total done on one or more numeric fields which appears in every record. This is a meaningless total, e.g., add the Telephone Numbers together for a number of Customers.

Limit check

Unlike range checks, data are checked for one limit only, upper OR lower, e.g., data should not be greater than 2 (<=2).

Logic check

Checks that an input does not yield a logical error, e.g., an input value should not be 0 when it will divide some other number somewhere in a program.

Presence check

Checks that important data is actually present and have not been missed out, e.g., customers may be required to have their telephone numbers listed.

Range check

Checks that the data is within a specified range of values, e.g., the month of a person's date of birth should lie between 1 and 12.

Referential integrity

In modern Relational database values in two tables can be linked through foreign key and primary key. If values in the primary key field are not constrained by database internal mechanism,[4] then they should be validated. Validation of the foreign key field checks that referencing table must always refer to a valid row in the referenced table.[5]

Spelling and grammar check

Looks for spelling and grammatical errors.

Uniqueness check

Checks that each value is unique. This can be applied to several fields (i.e. Address, First Name, Last Name).

Table look up check

A table look up check takes the entered data item and compares it to a valid list of entries that are stored in a database table.

Validation circumstances

In Add Category Activity

1) User should Add category name in Category name text box.

2)In Description User should enter the description.

3)User should select The mode of the amount transaction i.e , income or expense.

4)User should enter the amount in amount box.

Validation 1:

If user doesn`t enter Add category name in Category name text box, but enters description, selects transaction mode, enters amount, than pop-up will come up with message saying "please enter category name".

Validation 2:

If user enter Add category name in Category name text box, but doesn't enter description in description box, selects transaction mode, enters amount, than pop-up will come up with message saying "please enter description".

Validation 3:

If user enter Add category name in Category name text box, enters description in description box, selects transaction mode, but doesn't enters amount, than pop-up will come up with message saying "Amount cannot be nil..Enter the amount".

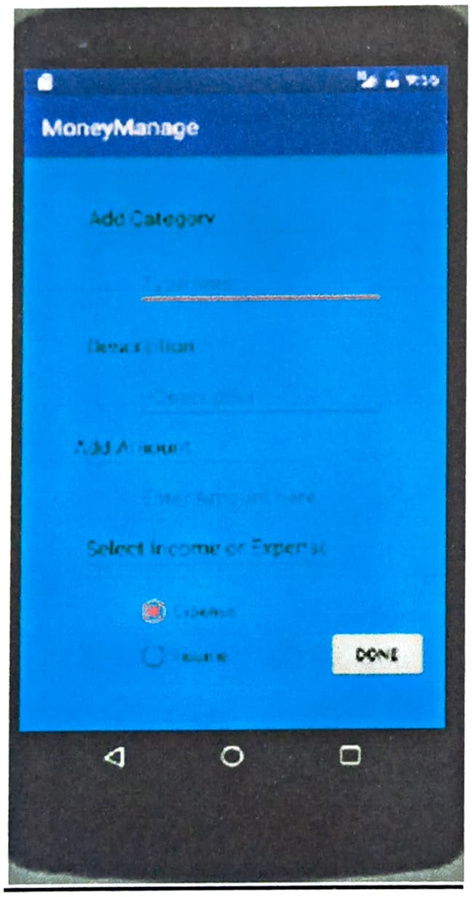
10.System Integration And test report

System integration (SI) is an IT or engineering process or phase concerned with joining different subsystems or components as one large system. It ensures that each integrated subsystem functions as required.

System Integration Testing(SIT) is a black box testing technique that evaluates the system's compliance against specified requirements. System Integration Testing is usually performed on subset of system while system testing is performed on a complete system and is preceded by the user acceptance test (UAT).

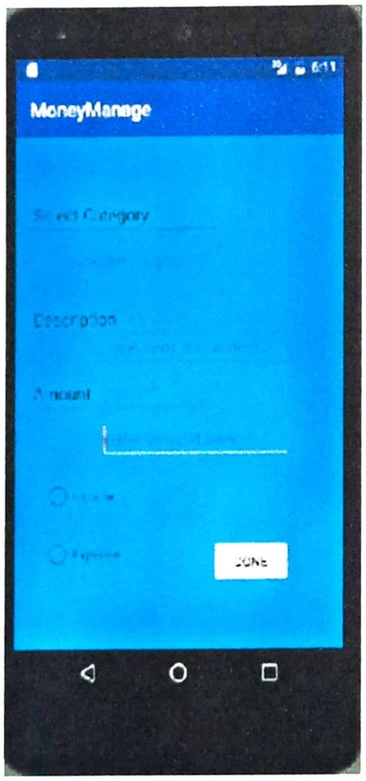
Add Category:

1) User must enter Category name.
2) User must enter Description of the category.
3) User must select mode of the transaction.
4) Enter amount of the transaction.
5) Click Done button.

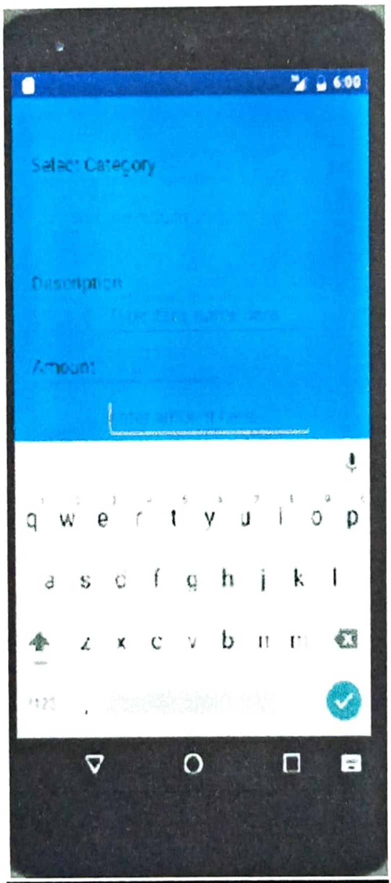


Select Category

1) User can select the category already existing in application.
2) Enter the description of the category.
3) Select the mode of a transaction.
4) Enter the amount.
5) Click on Done.

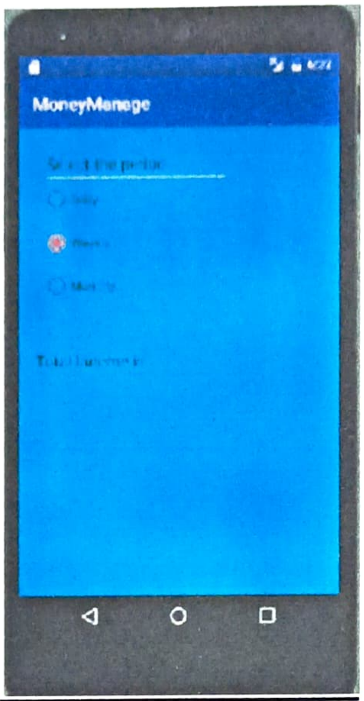


Select Category:



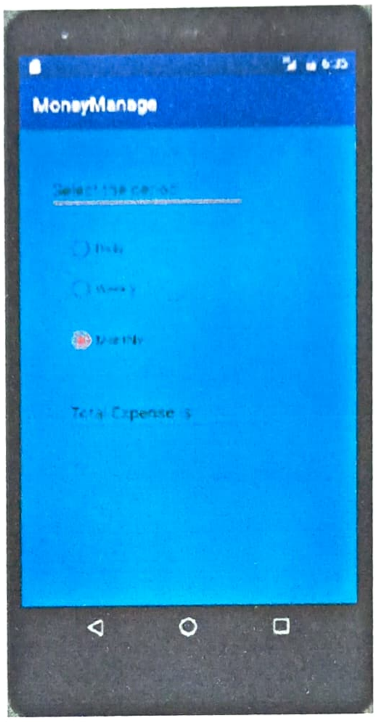
View Total Income

1) User will have to select the period to view the total income



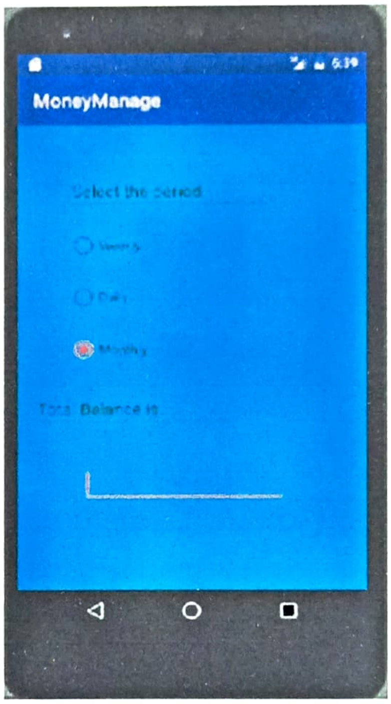
View Total Expense

1) User will have to select the period of which he want to see the expense.



Total Balance

1) User will have to select the period of which he wants to see the balance.



MoneyManage

Select the period

Weekly

Daily

[✓ Monthly]

Total Balance is

Datebase

Categories:-

Field Name	**Type**	**Size**	**Key**
Category_ID	INT	5	Primary
Date	DATE		
Category_Name	TEXT	30	Foreign
Category_Description	TEXT	50	

Transaction

Field Name	Type	Size
Category_ID	INT	5
Date	DATE	
Category_Name	TEXT	30
Income	INT	15
Expenses	INT	15
Balance	INT	20

11.User Manual

1)Open application named "Money Manage".

2)1st page will contain:

- Today's date with scrolling buttons to navigate within the dates

  -User will get a page opened with today's date and scrolling arrow buttons with it to navigate with date.

  -User will have to select the date in which he has to enter data for income or expense.
- Select Category (button):

  -Click on Select Category Button.

  -In this user will have to select the category already present in the from the select category drop down box.

  -Enter the description in description text area.

  -Select whether the amount to be entered is income or expense using radio buttons namely income and expense.

  -Enter the Amount in Amount area.

  -Click on DONE button.
- Add Category (button):

  -click on Add Category Button.

  -user will have to add category if the category in not present in select category list.

  -Enter the description in description text area.

  -Enter the Amount in Amount area.

  -Click on DONE button.
- View Total Income(button)

  -Select the category period of which user want to see the income.

  -And the Total Income Amount will be viewed in the amount area.
- View Total Expense(button)

  -Select the category period of which user want to see the income.

  -And the Total Expense Amount will be viewed in the amount area.
- Total Balance(button)

  -Select the category period of which user want to see the income.

  -And the Total Balance Amount will be viewed in the amount area.

12.Future Enhancement

1)Setting reminder

2)Allowing user to set his budget.

3)Sending alerts through popup if user exceeding budget.

4)Viewing graph as per user interaction, Such as:

a)Viewing graph of expenses as per user defined category

b) Viewing graph of income as per user defined category

I.References

1. www.tutorialspoint.com
2. www.stackoverflow.com

II.Bibliography

Books:

1) **Android Studio Application Development**
2) **Android studio Essential Previews**
3) **Professional Android Application development**

Annexure III

WORK RECORD/DIARY

Name of the College :Goa Multi-Faculty College

Name of the Candidate : Rekha Paranjape

Course : BCA

Year : 2016-2017

Title of the Project: <u>MoneyMangement Application</u>

Library/Laboratory Fieldwork	Description of work	Date	Time spent	Signature of Authority	Counter Signature of Guide & Date
Library	Reading books	05/11/2016	2 hours	[illegible]	
Library	Reading Books	19/11/2016	3 hours	[illegible]	
Library	Reading Books	03/12/2016	1 hours	[illegible]	
Laboratory	Project work	10/12/2016	4 hours	[illegible]	[illegible]
Laboratory	Project work	17/12/2016	3 hours	[illegible]	
Laboratory	Project work	27/12/2016	4 hours	[illegible]	
Library	Reading books	28/12/2016	2 hours	[illegible]	
Library	Reading books	07/01/2017	3 hours	[illegible]	
Laboratory	Project work	14/01/2017	3 hours	[illegible]	
Laboratory	Project work	15/01/2017	4 hours	[illegible]	

1. Signature of the Student : Paranjape

2. Signature of the Guide : [illegible]

3. Signature of Project Coordinator :

Annexure III

WORK RECORD/DIARY

Name of the College :Goa Multi-Faculty College

Name of the Candidate : Anuja Jawadekar

Course : BCA

Year : 2016-2017

Title of the Project: <u>MoneyMangement Application</u>

Library/Laboratory Fieldwork	Description of work	Date	Time spent	Signature of Authority	Counter Signature of Guide & Date
Library	Reading books	05/11/2016	2 hours	[illegible]	
Library	Reading Books	19/11/2016	3 hours	[illegible]	
Library	Reading Books	03/12/2016	1 hours	[illegible]	
Laboratory	Project work	10/12/2016	4 hours	[illegible]	[illegible]
Laboratory	Project work	17/12/2016	3 hours	[illegible]	
Laboratory	Project work	27/12/2016	4 hours	[illegible]	
Library	Reading books	28/12/2016	2 hours	[illegible]	
Library	Reading books	07/01/2017	3 hours	[illegible]	
Laboratory	Project work	14/01/2017	3 hours	[illegible]	
Laboratory	Project work	15/01/2017	4 hours	[illegible]	

1. Signature of the Student : Jawadekar

2. Signature of the Guide : [illegible]

3. Signature of Project Coordinator :

Annexure III

WORK RECORD/DIARY

Name of the College :Goa Multi-Faculty College

Name of the Candidate : Vallabh Naik

Course : BCA

Year : 2016-2017

Title of the Project: <u>MoneyMangement Application</u>

Library/Laboratory Fieldwork	Description of work	Date	Time spent	Signature of Authority	Counter Signature of Guide & Date
Library	Reading books	05/11/2016	2 hours	[illegible]	
Library	Reading Books	19/11/2016	3 hours	[illegible]	
Library	Reading Books	03/12/2016	1 hours	[illegible]	
Laboratory	Project work	10/12/2016	4 hours	[illegible]	
Laboratory	Project work	17/12/2016	3 hours	[illegible]	[illegible]
Laboratory	Project work	27/12/2016	4 hours	[illegible]	
Library	Reading books	28/12/2016	2 hours	[illegible]	
Library	Reading books	07/01/2017	3 hours	[illegible]	
Laboratory	Project work	14/01/2017	3 hours	[illegible]	
Laboratory	Project work	15/01/2017	4 hours	[illegible]	

1. Signature of the Student : [illegible]

2. Signature of the Guide : [illegible]

3. Signature of Project Coordinator :

Annexure III

WORK RECORD/DIARY

Name of the College :Goa Multi-Faculty College

Name of the Candidate : Mayur Pilyenkar

Course : BCA

Year : 2016-2017

Title of the Project: MoneyMangement Application

Library/Laboratory Fieldwork	Description of work	Date	Time spent	Signature of Authority	Counter Signature of Guide & Date
Library	Reading books	05/11/2016	2 hours	[illegible]	
Library	Reading Books	19/11/2016	3 hours	[illegible]	
Library	Reading Books	03/12/2016	1 hours	[illegible]	
Laboratory	Project work	10/12/2016	4 hours	[illegible]	
Laboratory	Project work	17/12/2016	3 hours	[illegible]	[illegible]
Laboratory	Project work	27/12/2016	4 hours	[illegible]	
Library	Reading books	28/12/2016	2 hours	[illegible]	
Library	Reading books	07/01/2017	3 hours	[illegible]	
Laboratory	Project work	14/01/2017	3 hours	[illegible]	
Laboratory	Project work	15/01/2017	4 hours	[illegible]	

1. Signature of the Student : [illegible]

2. Signature of the Guide : [illegible]

3. Signature of Project Coordinator :

Annexure VII

PROJECT GANTT CHART

Task	Duration(Start and end dates of each task in a calendar format)
Analysis	5/11/2016 to 19/11/2016
Design	20/11/2016 to 3/12/2016
Implementation	4/12/2016 to 7/01/2017
Testing	8/01/2017 to15/01/2017

Name & Signature of project Guide