BACHAT

THE MONEY TRACKER.

⭐Problem Statement:

- We don't have a decent software for keeping track of our spending, we still write them down on paper.

- We still need to go to the bank to print our transactions on paper because there is no special app for checking our all transactions .

⭐Introduction:

Who doesn't have a New Year's financial resolution? Many of us have financial objectives, whether it's to create more money or to better manage the money we already have — but regardless of your objectives, you'll almost likely require some baseline data. While understanding your expenses may appear to be straightforward, they include some of the most important information about where your money goes. **Keeping track of your expenses is a straightforward chore that can provide you with a set of data.**

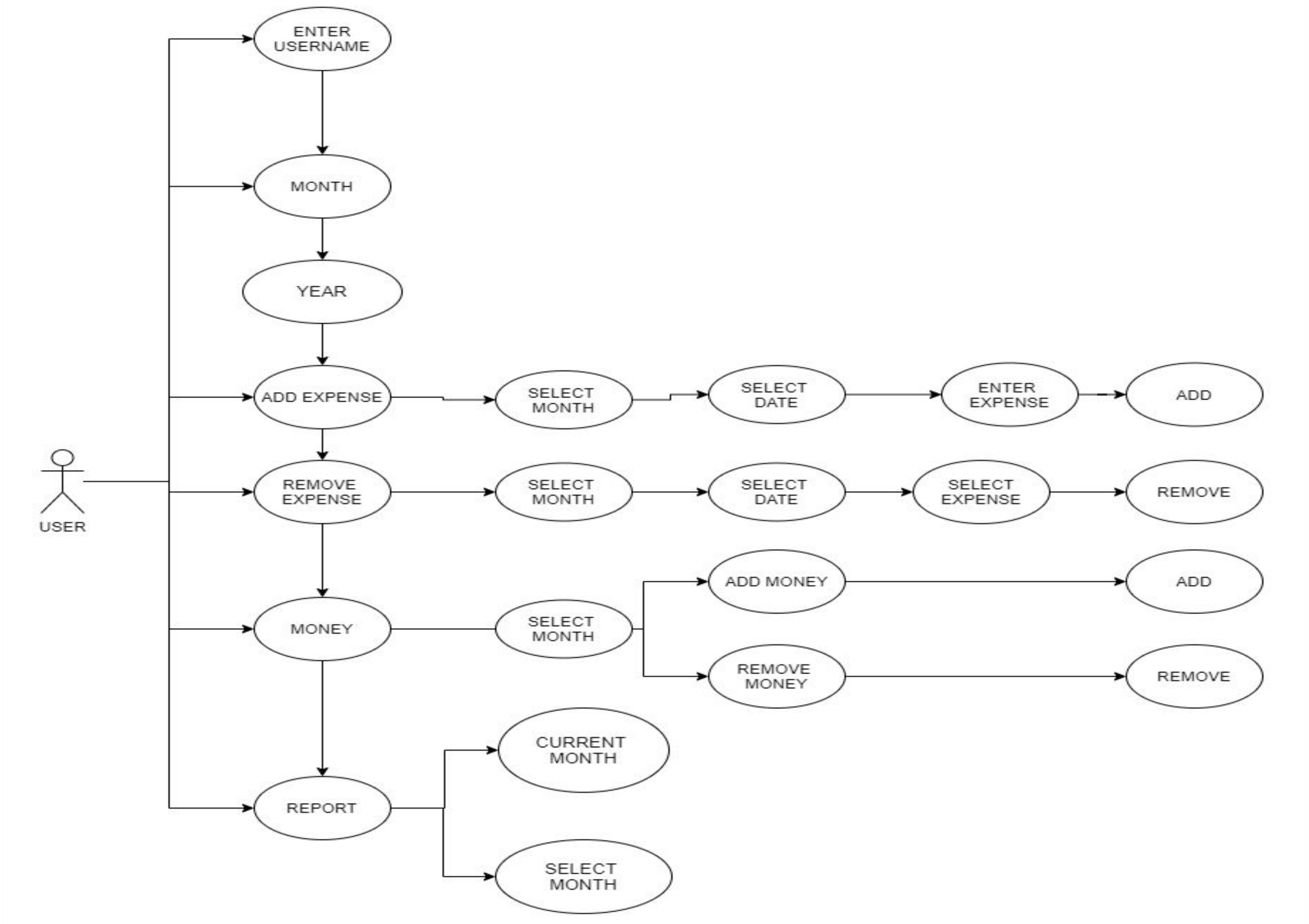
⭐Use Case Diagram:

- The **Use-Case** diagram depicts the interactions between the system administrator, the bet creator, and the system participants.

- It is a graphical representation of the user's potential interactions with the system.

- This graphic depicts all of the system's stakeholders' interactions as well as how the system functions.

use case diagram



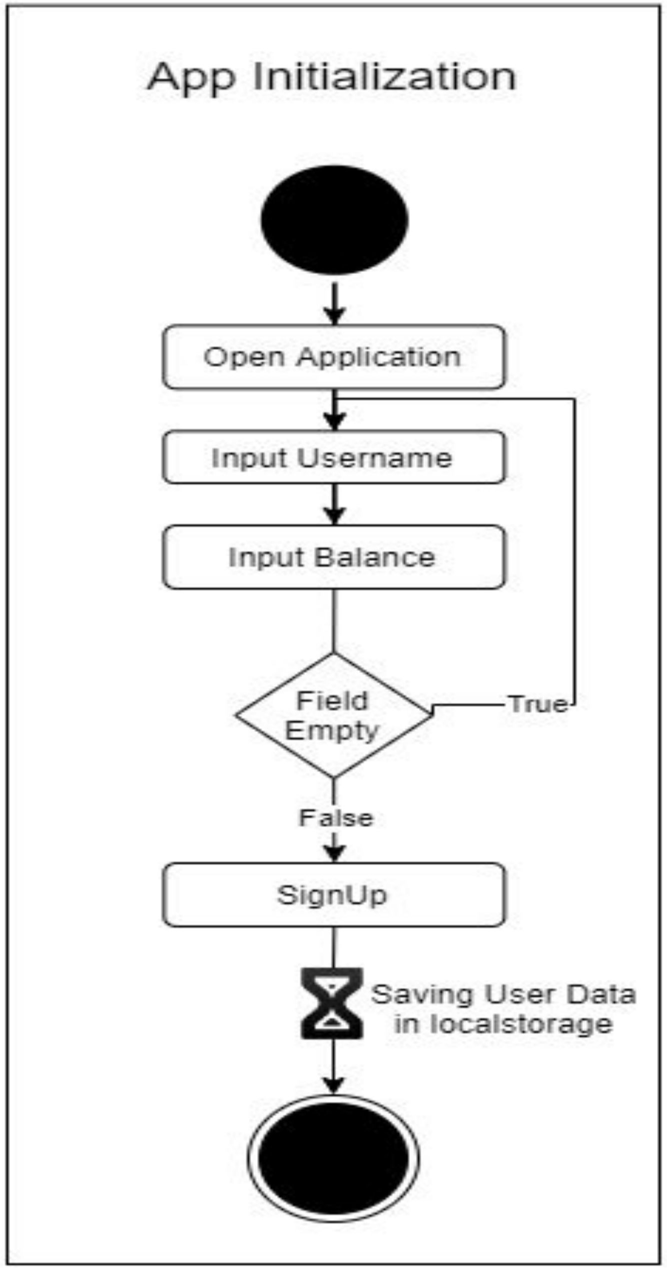
⭐Activity Diagram:

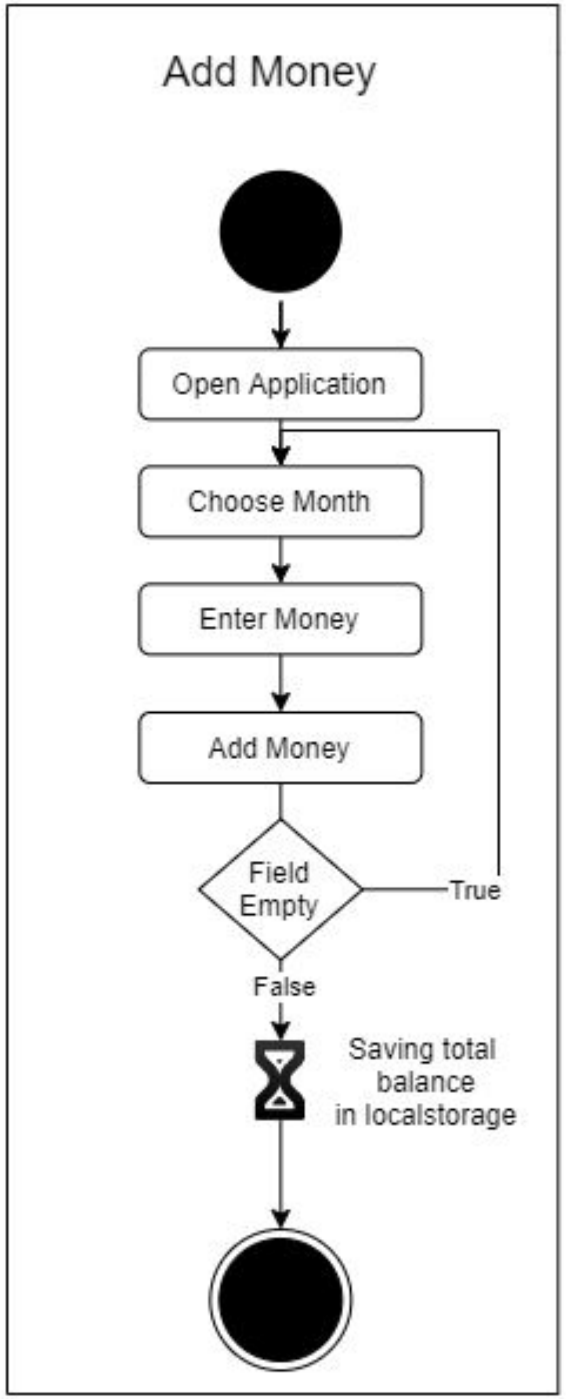
- Activity diagram is another important diagram in UML to describe the dynamic aspects of the system.

- Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.

- The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc

Activity Diagram





Add Money

Open Application

Choose Month

Enter Money

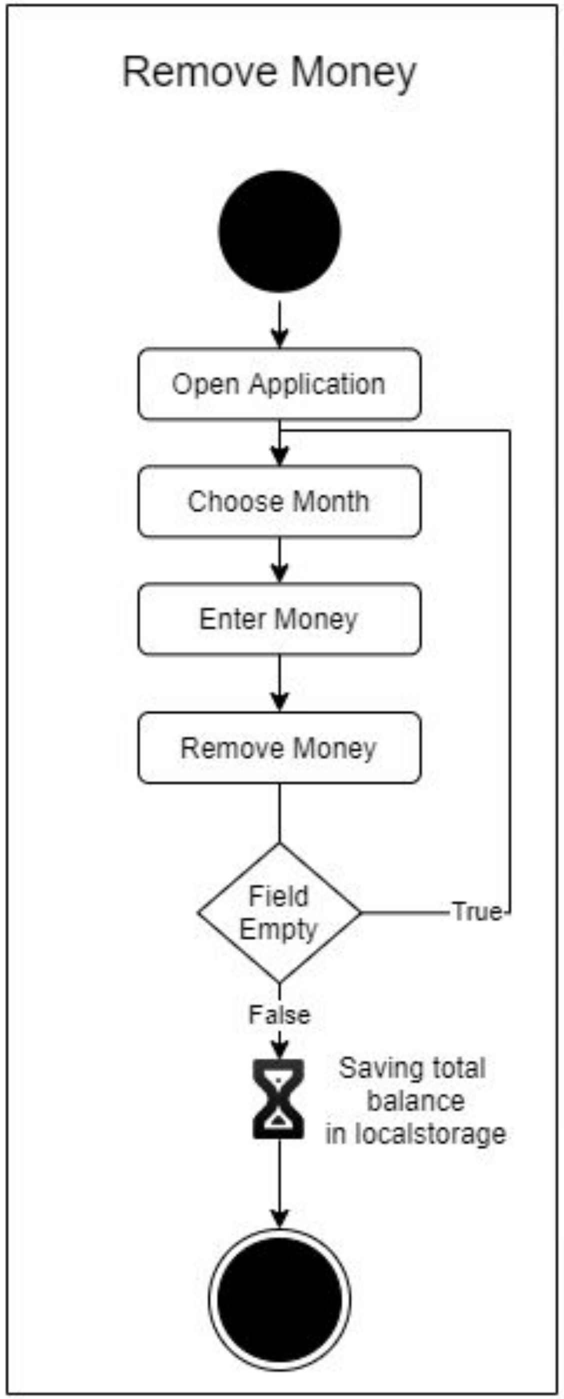
Add Money

Field Empty

True

False

Saving total balance in localstorage



Remove Money

Open Application

Choose Month

Enter Money

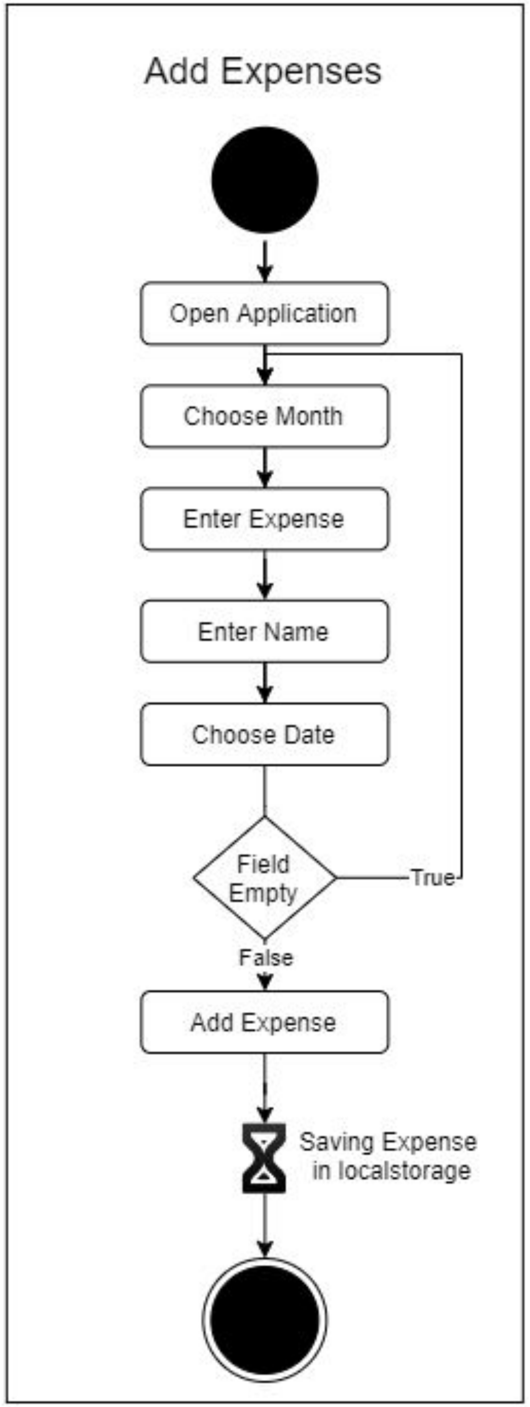
Remove Money

Field Empty

True

False

Saving total balance in localstorage



Add Expenses

Open Application

Choose Month

Enter Expense

Enter Name

Choose Date

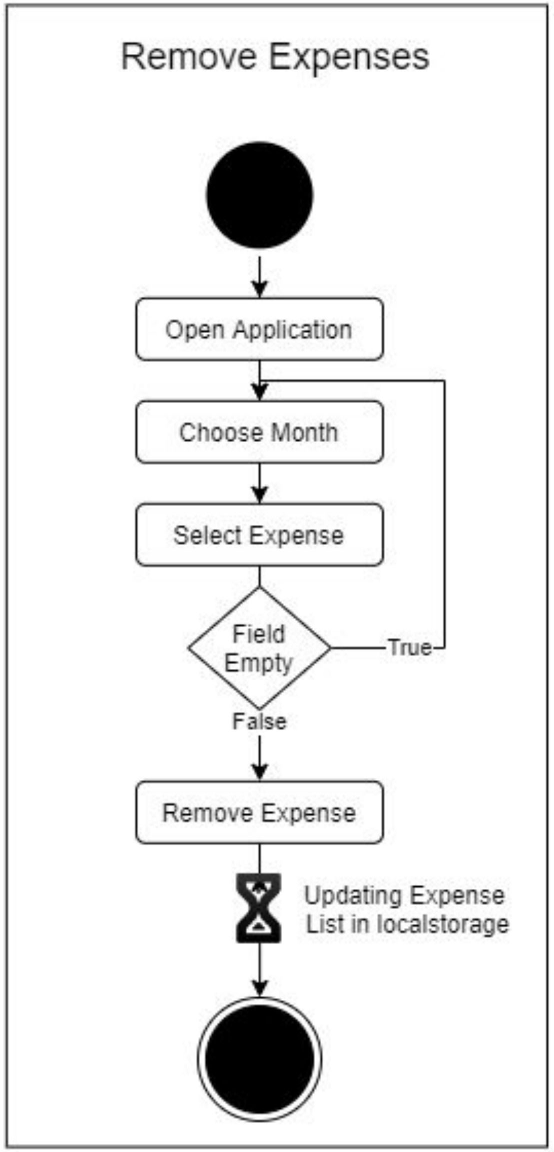
Field Empty

True

False

Add Expense

Saving Expense in localstorage



Remove Expenses

Open Application

Choose Month

Select Expense

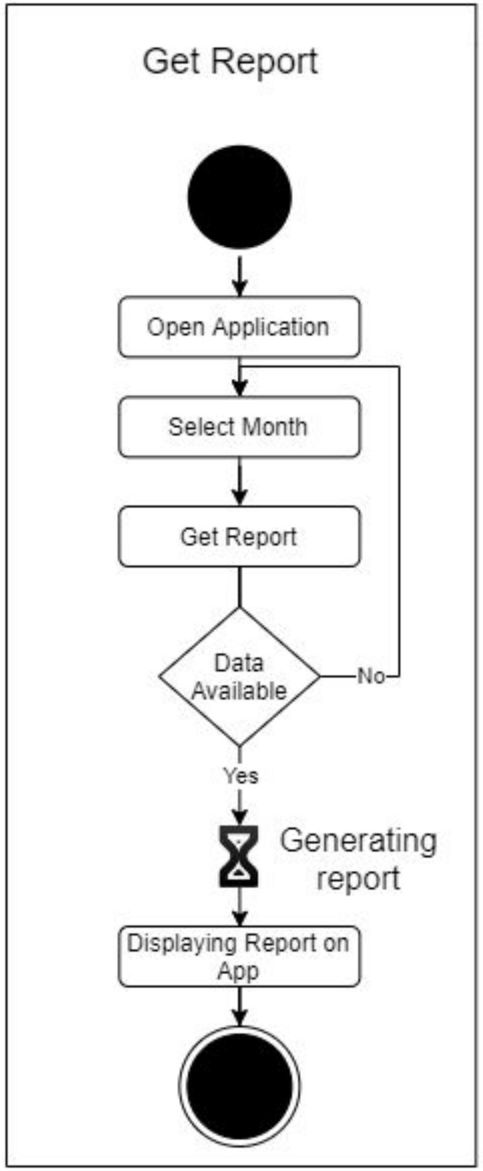
Field Empty

True

False

Remove Expense

Updating Expense List in localstorage



Get Report

Open Application

Select Month

Get Report

Data Available

No

Yes

Generating report

Displaying Report on App

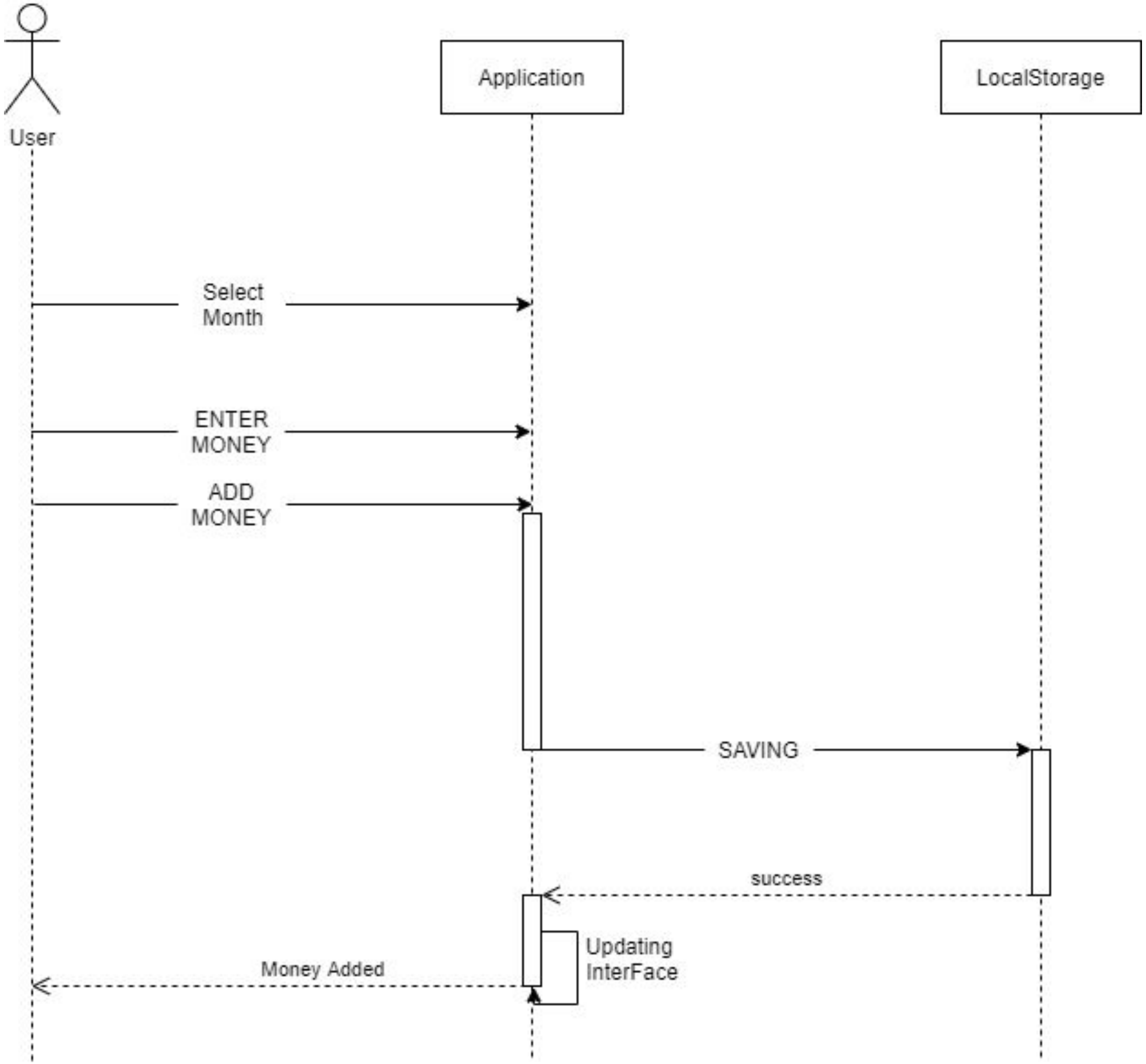
⭐Sequence Diagram:

- A sequence diagram, in the context of UML, represents object collaboration and is used to define event sequences between objects for a certain outcome. A sequence diagram is an essential component used in processes related to analysis, design and documentation.

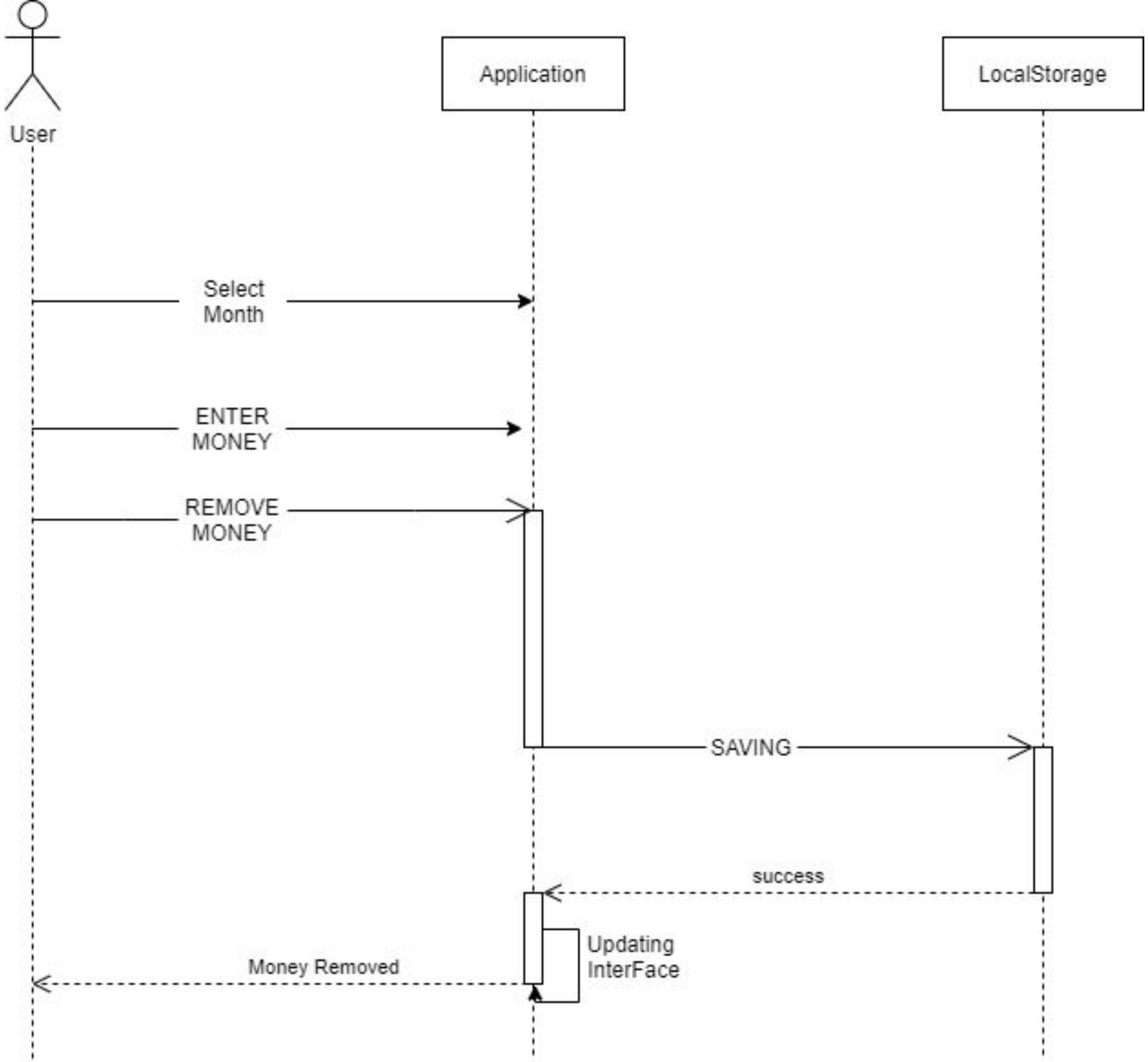
  A sequence diagram is also known as a timing diagram, event diagram and event scenario.

Add Money:-

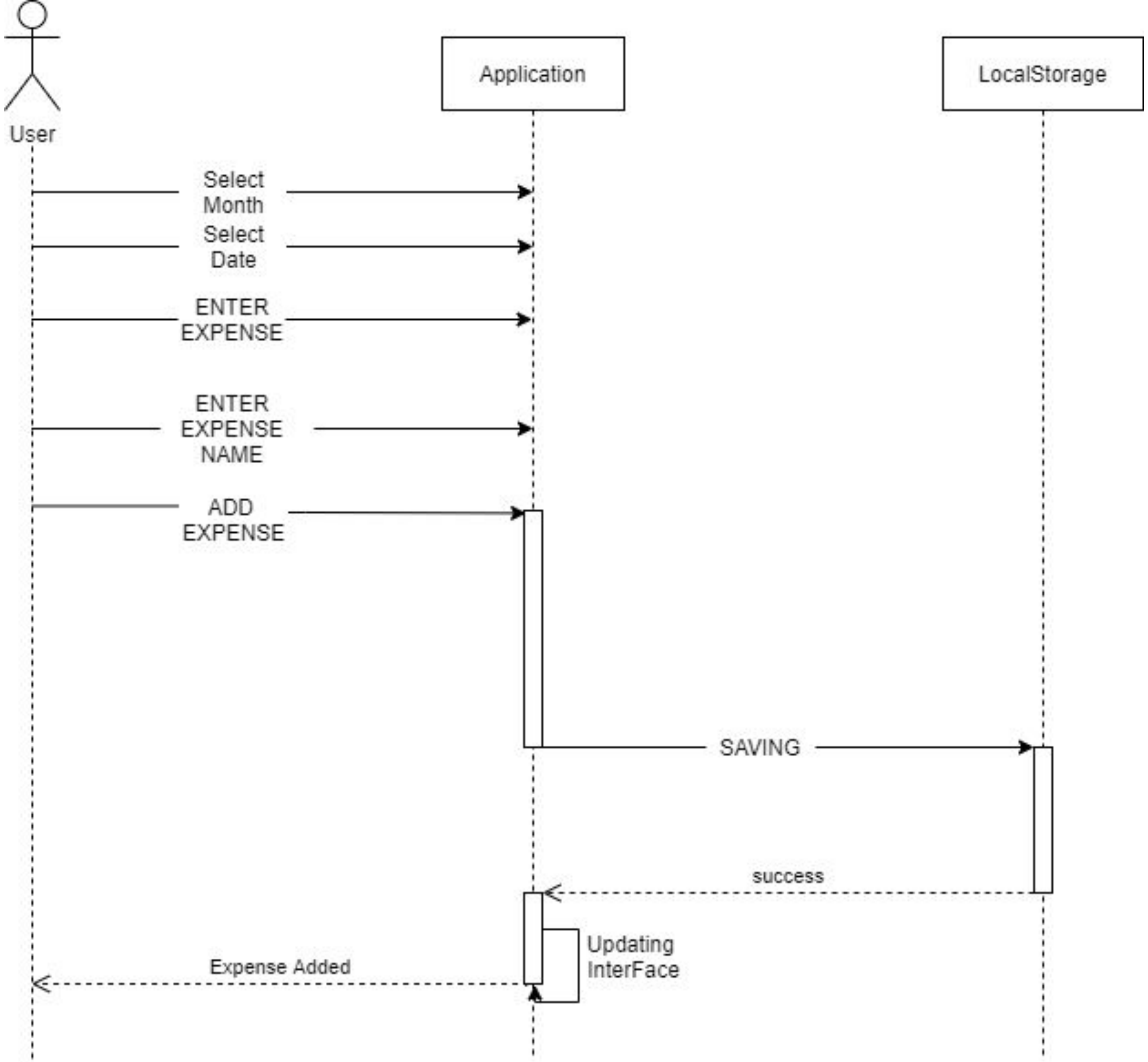
Sequence Diagram



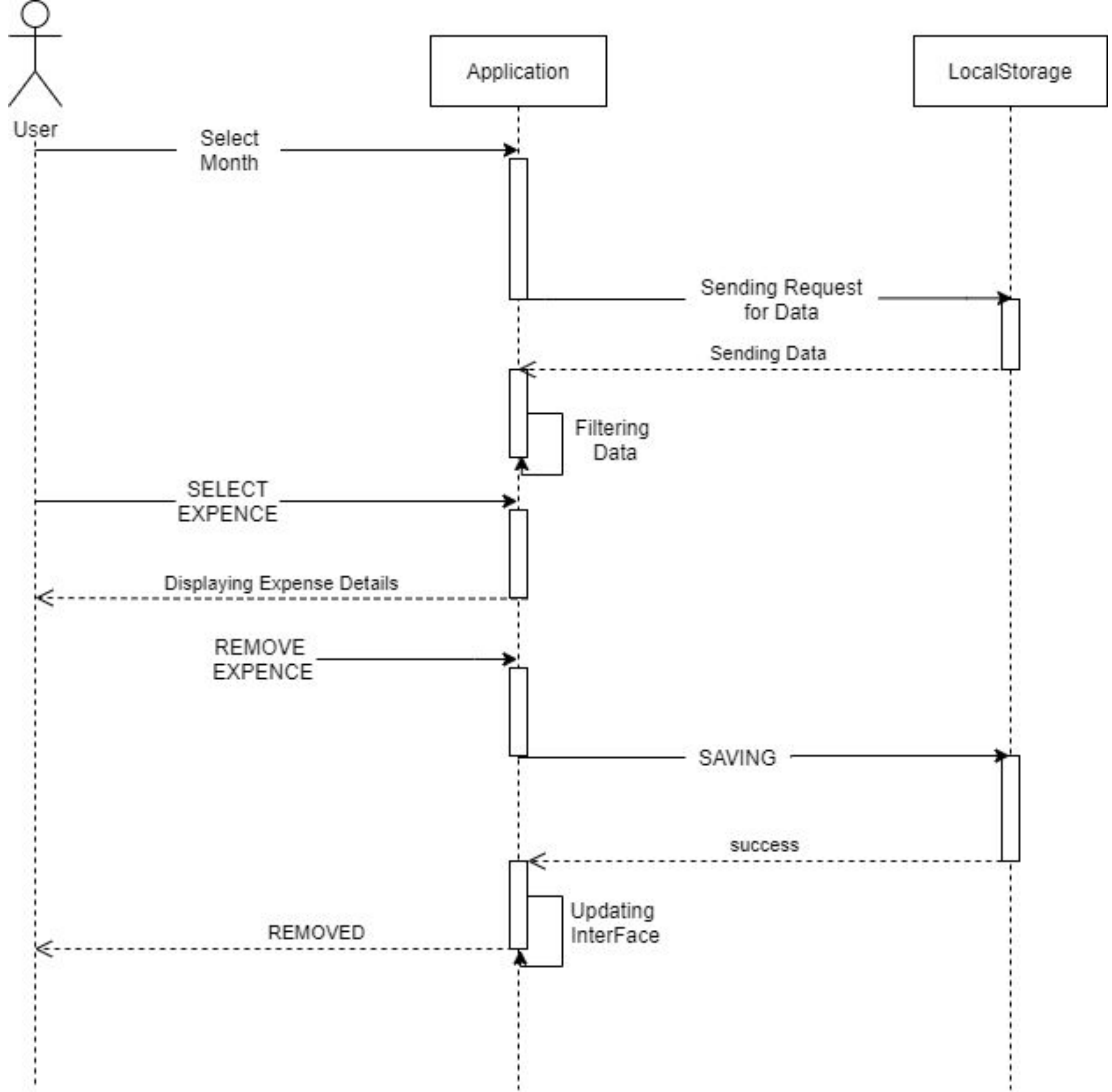
Remove Money:-



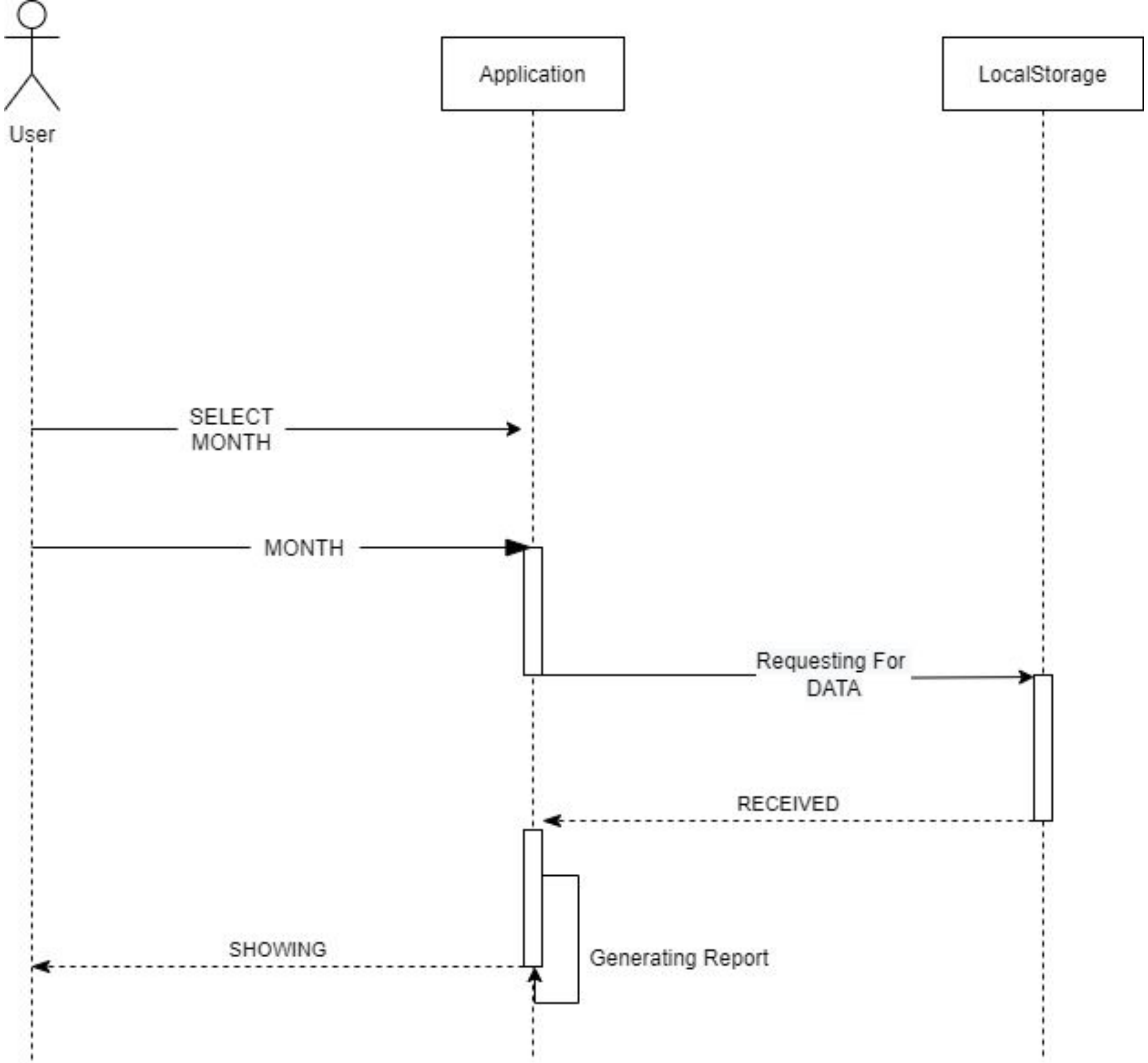
Add Expense:-


User
Application
LocalStorage
Select Month
Select Date
ENTER EXPENSE
ENTER EXPENSE NAME
ADD EXPENSE
SAVING
success
Updating InterFace
Expense Added

Remove Expense:-



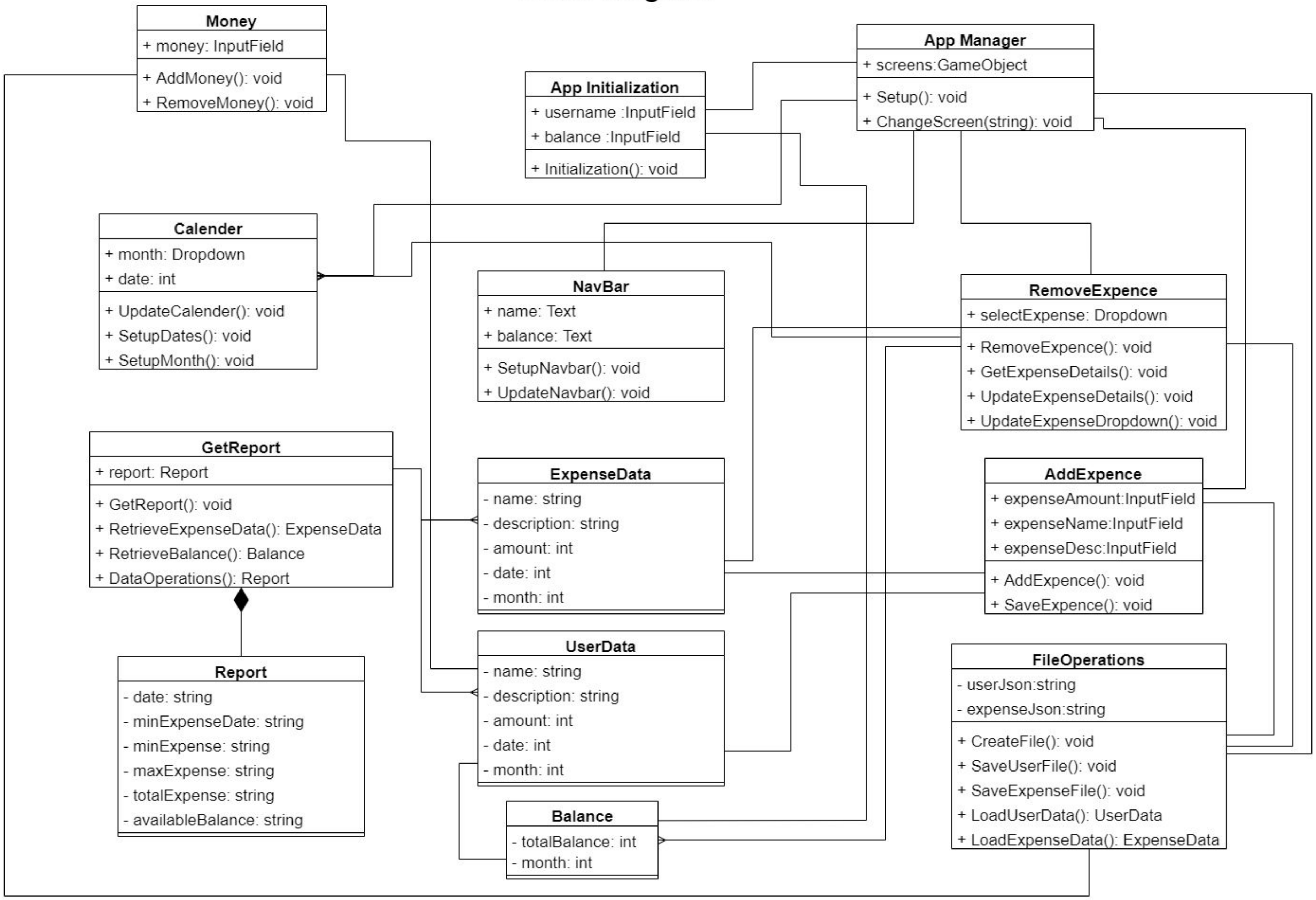
Report:-


User
Application
LocalStorage
SELECT MONTH
MONTH
Requesting For DATA
RECEIVED
Generating Report
SHOWING

⭐Class Diagram:

- Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.
- Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.
- Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

Class Diagram



Money
- + money: InputField
- + AddMoney(): void
- + RemoveMoney(): void

App Initialization
- + username :InputField
- + balance :InputField
- + Initialization(): void

App Manager
- + screens:GameObject
- + Setup(): void
- + ChangeScreen(string): void

Calender
- + month: Dropdown
- + date: int
- + UpdateCalender(): void
- + SetupDates(): void
- + SetupMonth(): void

NavBar
- + name: Text
- + balance: Text
- + SetupNavbar(): void
- + UpdateNavbar(): void

RemoveExpence
- + selectExpense: Dropdown
- + RemoveExpence(): void
- + GetExpenseDetails(): void
- + UpdateExpenseDetails(): void
- + UpdateExpenseDropdown(): void

GetReport
- + report: Report
- + GetReport(): void
- + RetrieveExpenseData(): ExpenseData
- + RetrieveBalance(): Balance
- + DataOperations(): Report

ExpenseData
- - name: string
- - description: string
- - amount: int
- - date: int
- - month: int

AddExpence
- + expenseAmount:InputField
- + expenseName:InputField
- + expenseDesc:InputField
- + AddExpence(): void
- + SaveExpence(): void

Report
- - date: string
- - minExpenseDate: string
- - minExpense: string
- - maxExpense: string
- - totalExpense: string
- - availableBalance: string

UserData
- - name: string
- - description: string
- - amount: int
- - date: int
- - month: int

FileOperations
- - userJson:string
- - expenseJson:string
- + CreateFile(): void
- + SaveUserFile(): void
- + SaveExpenseFile(): void
- + LoadUserData(): UserData
- + LoadExpenseData(): ExpenseData

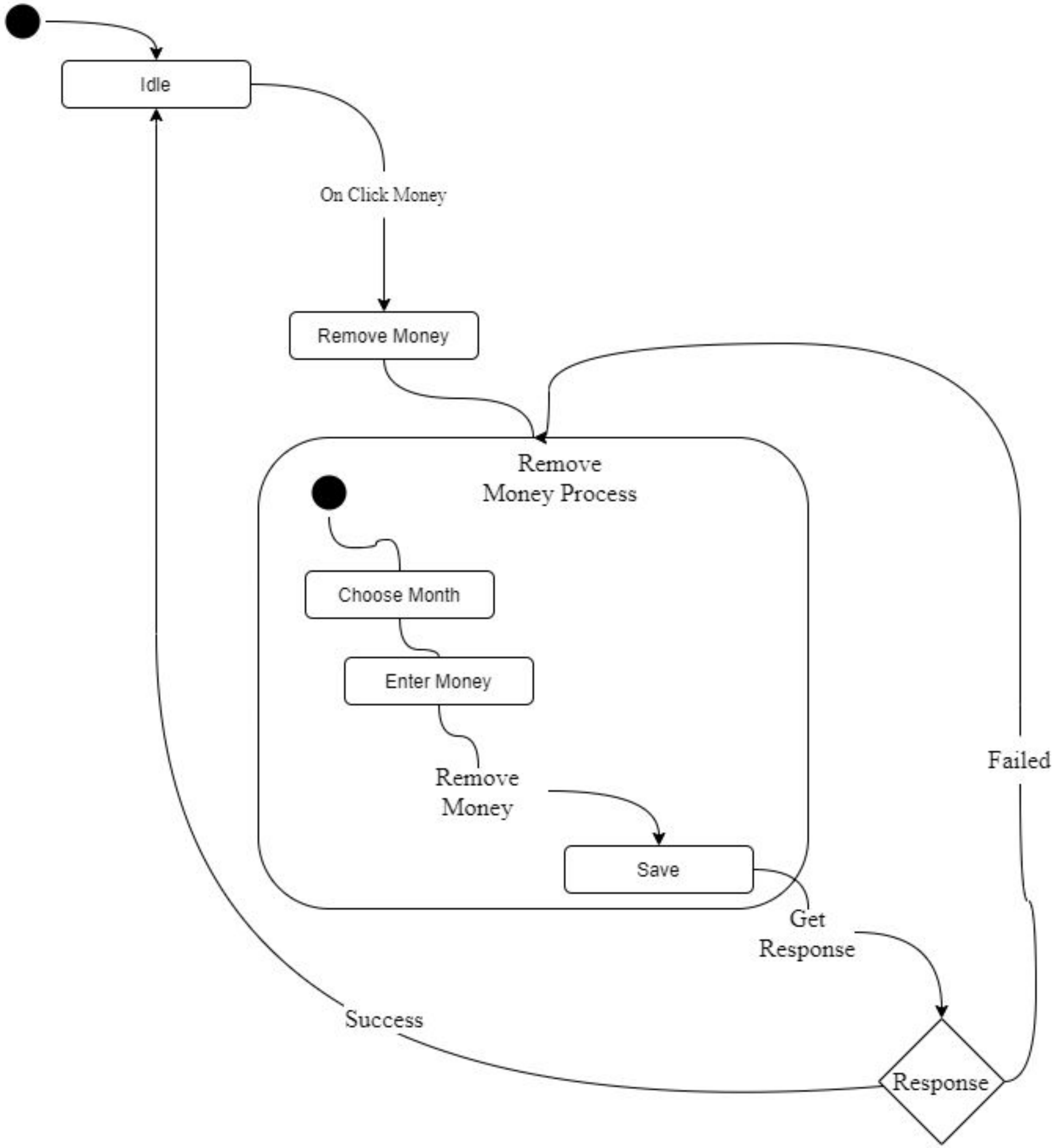
Balance
- - totalBalance: int
- - month: int

⭐State Diagram:

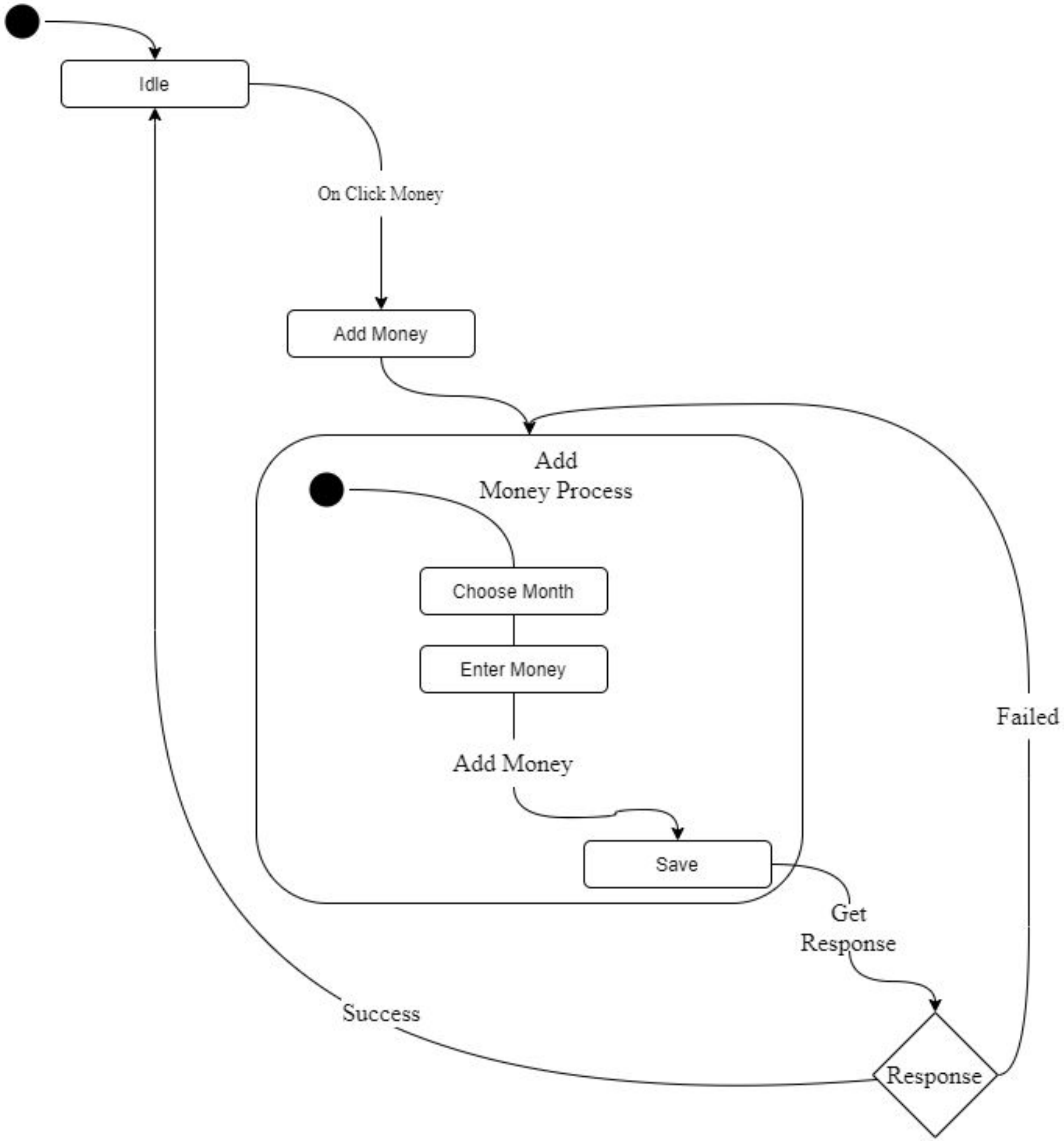
- A state diagram is a diagram used in computer science to describe the behavior of a system considering all the possible states of an object when an event occurs. This behavior is represented and analyzed in a series of events that occur in one or more possible states. Each diagram represents objects and tracks the various states of these objects throughout the system.

Remove Money:-

State Diagram

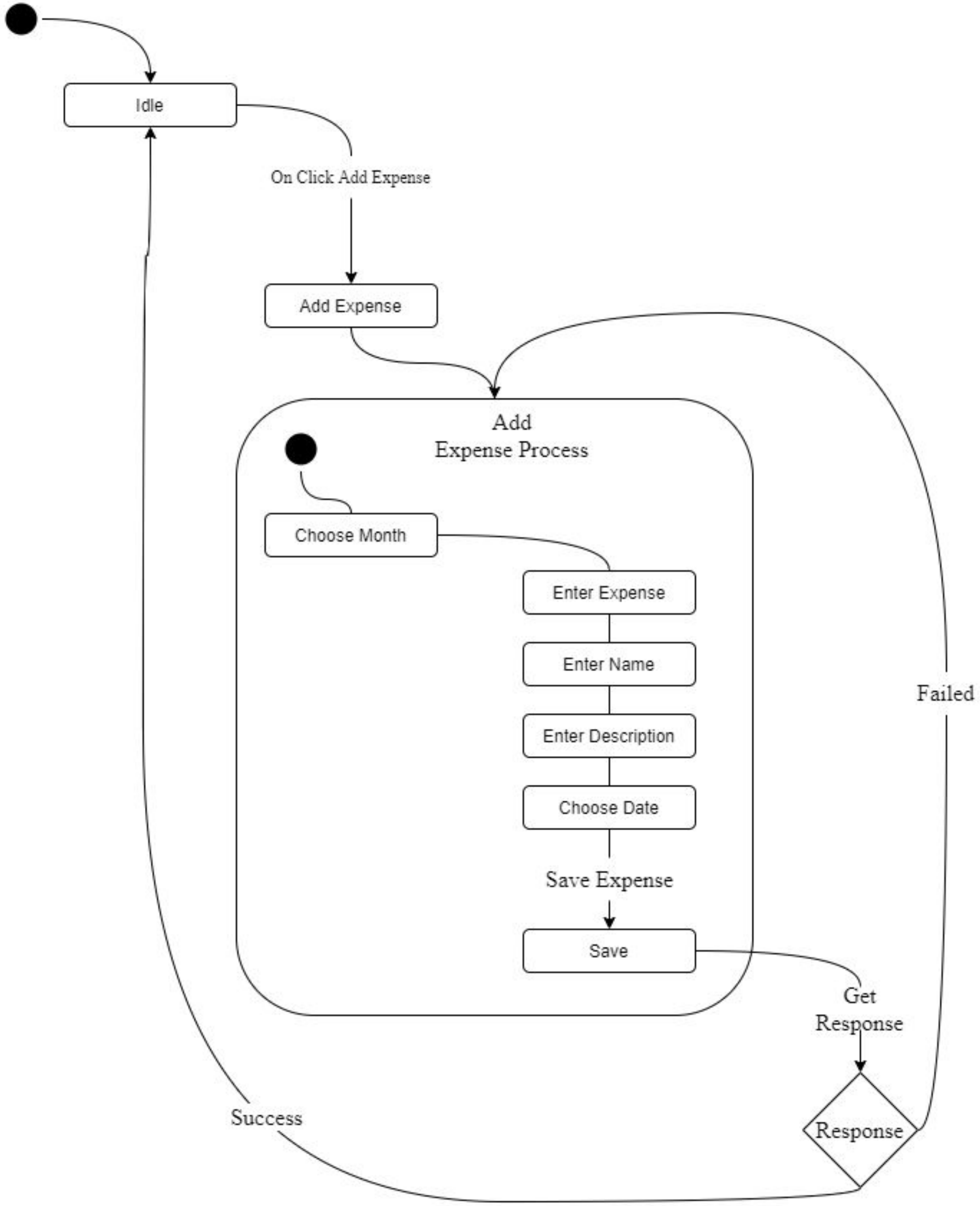


Add Money:-

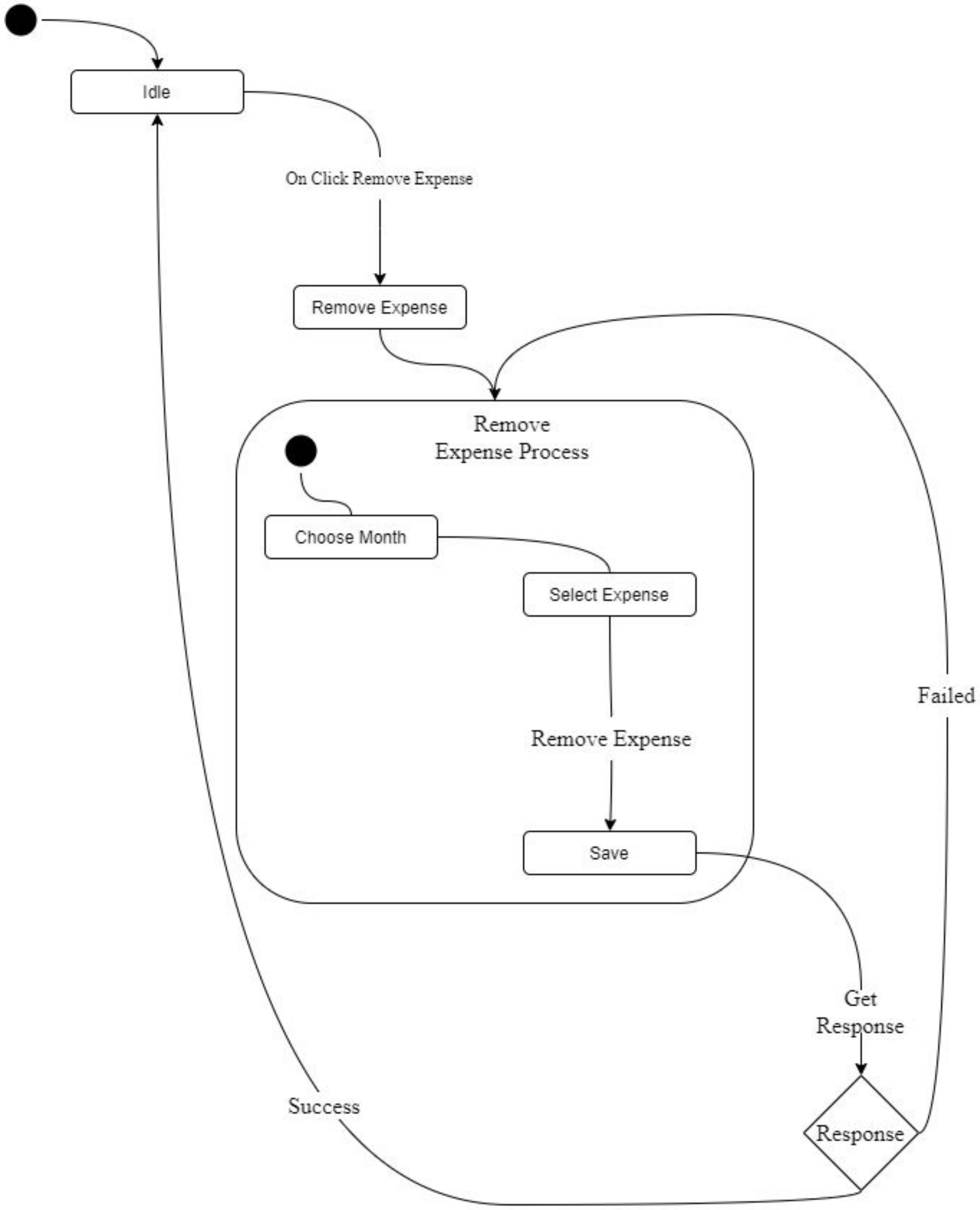


Idle
On Click Money
Add Money
Add Money Process
Choose Month
Enter Money
Add Money
Save
Get Response
Failed
Success
Response

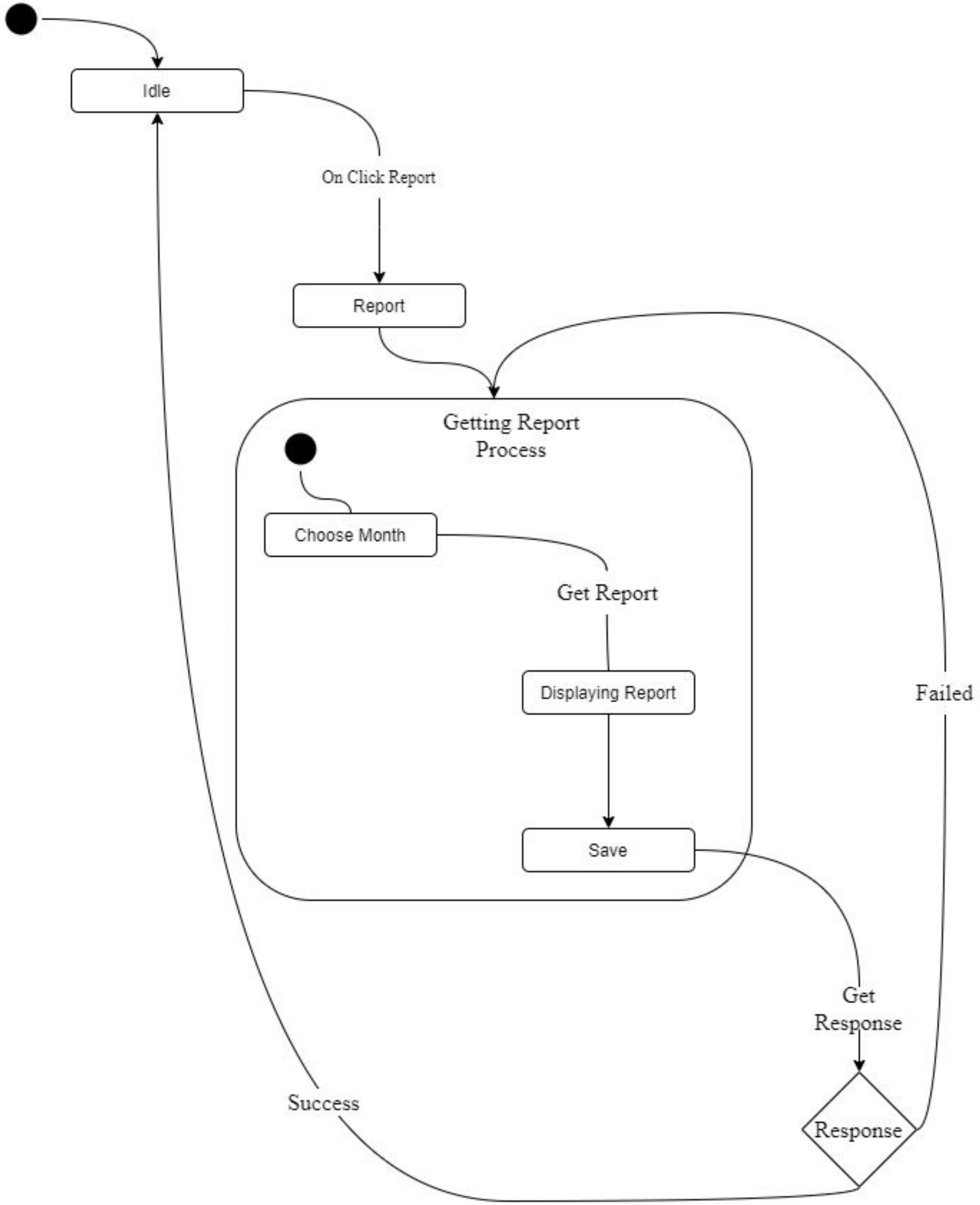
Add Expense:-


Idle
On Click Add Expense
Add Expense
Add
Expense Process
Choose Month
Enter Expense
Enter Name
Enter Description
Choose Date
Save Expense
Save
Get
Response
Response
Failed
Success

Remove Expense:-



Report:-


Idle
On Click Report
Report
Getting Report Process
Choose Month
Get Report
Displaying Report
Save
Get Response
Response
Failed
Success

⭐ER-Diagram:

- ER-modeling is a data modeling method used in software engineering to produce a conceptual data model of an information system. Diagrams created using this ER-modeling method are called Entity-Relationship Diagrams or ER diagrams or ERDs.

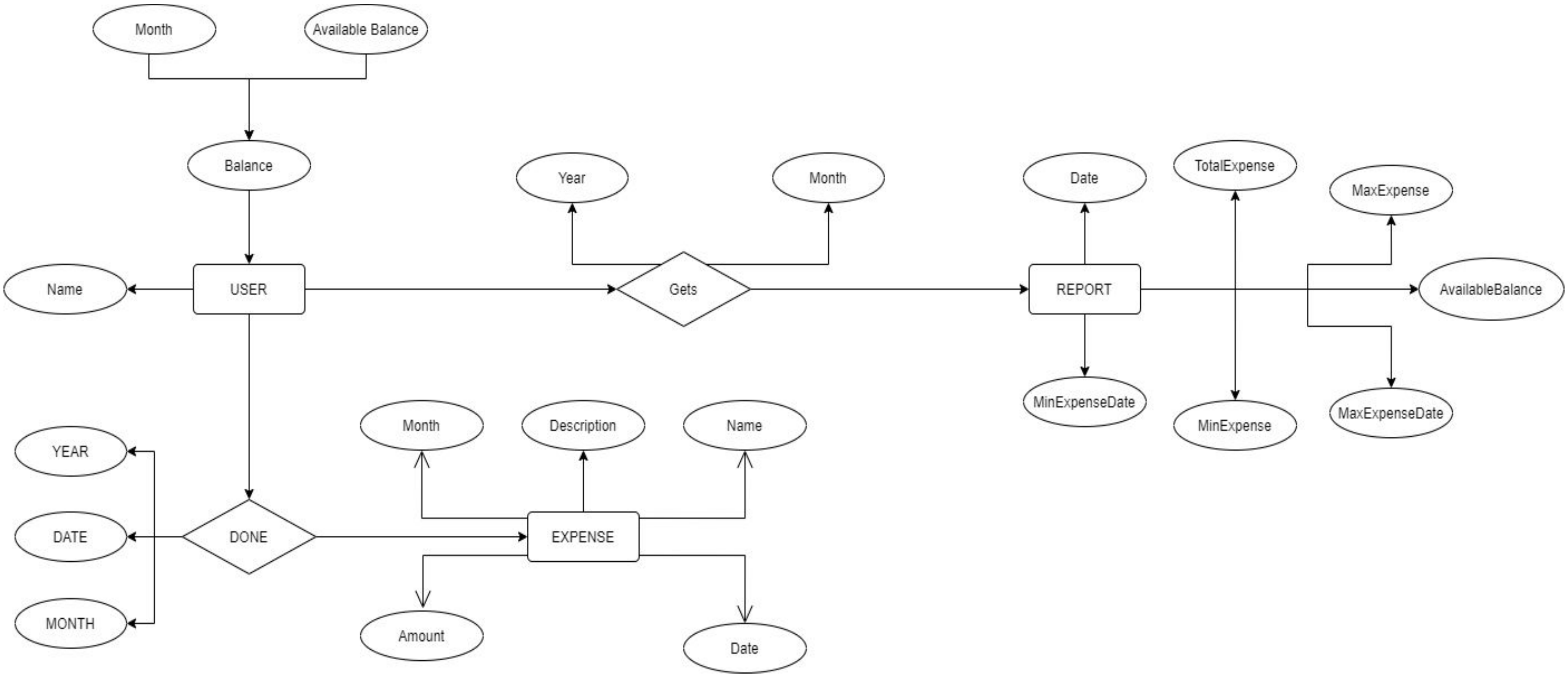
- Purpose of ERD

- The database analyst gains a better understanding of the data to be contained in the database through the step of constructing the ERD.

- The ERD serves as a documentation tool.

- Finally, the ERD is used to connect the logical structure of the database to users. In particular, the ERD effectively communicates the logic of the database to users.

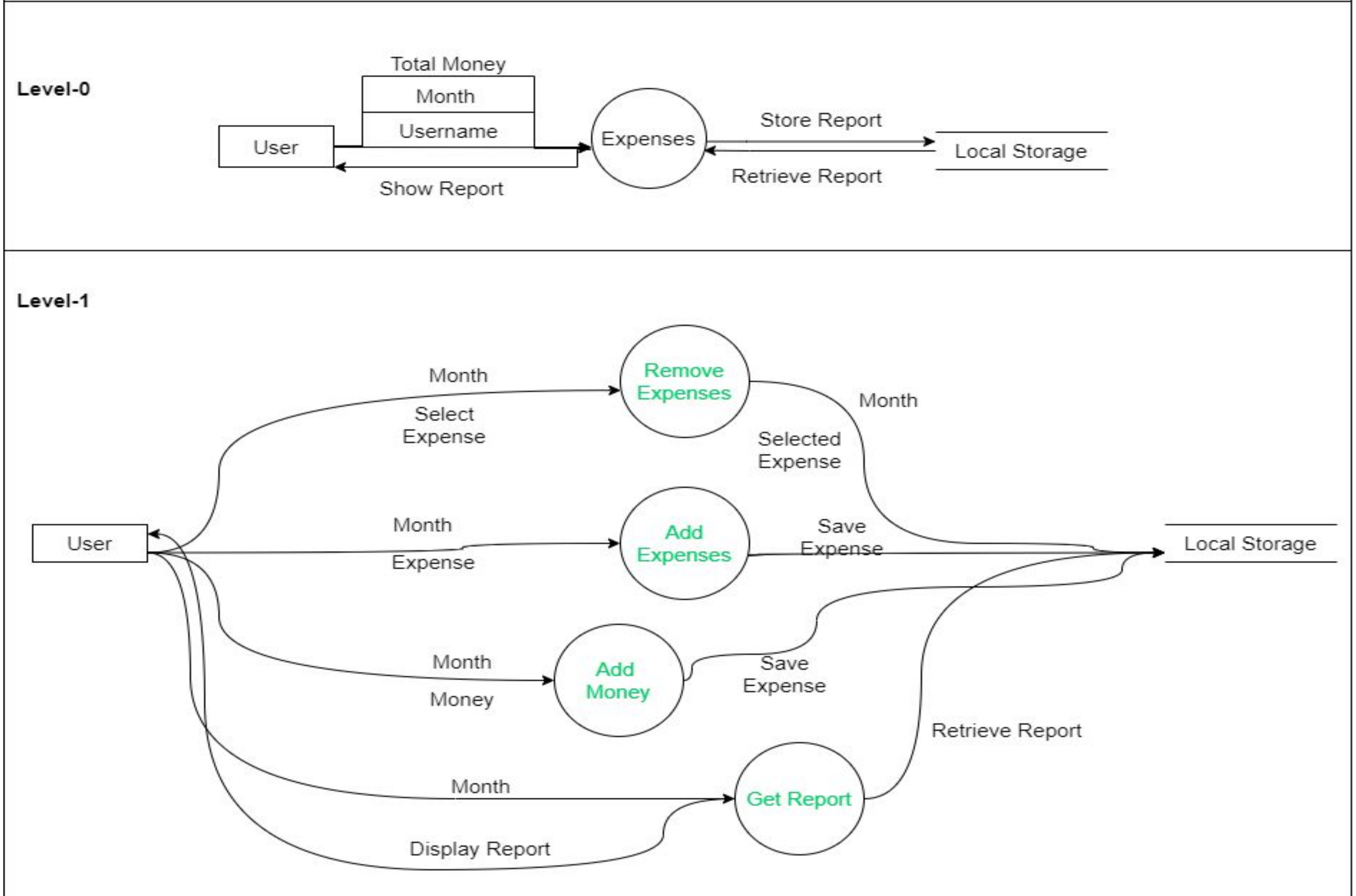
ER-Diagram


Month
Available Balance
Balance
Year
Month
Date
TotalExpense
MaxExpense
Name
USER
Gets
REPORT
AvailableBalance
MinExpenseDate
MinExpense
MaxExpenseDate
Month
Description
Name
YEAR
DATE
DONE
EXPENSE
MONTH
Amount
Date

⭐Data Flow Diagram:

- A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.
- It shows how data enters and leaves the system, what changes the information, and where data is stored.
- The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

DataFlow Diagram



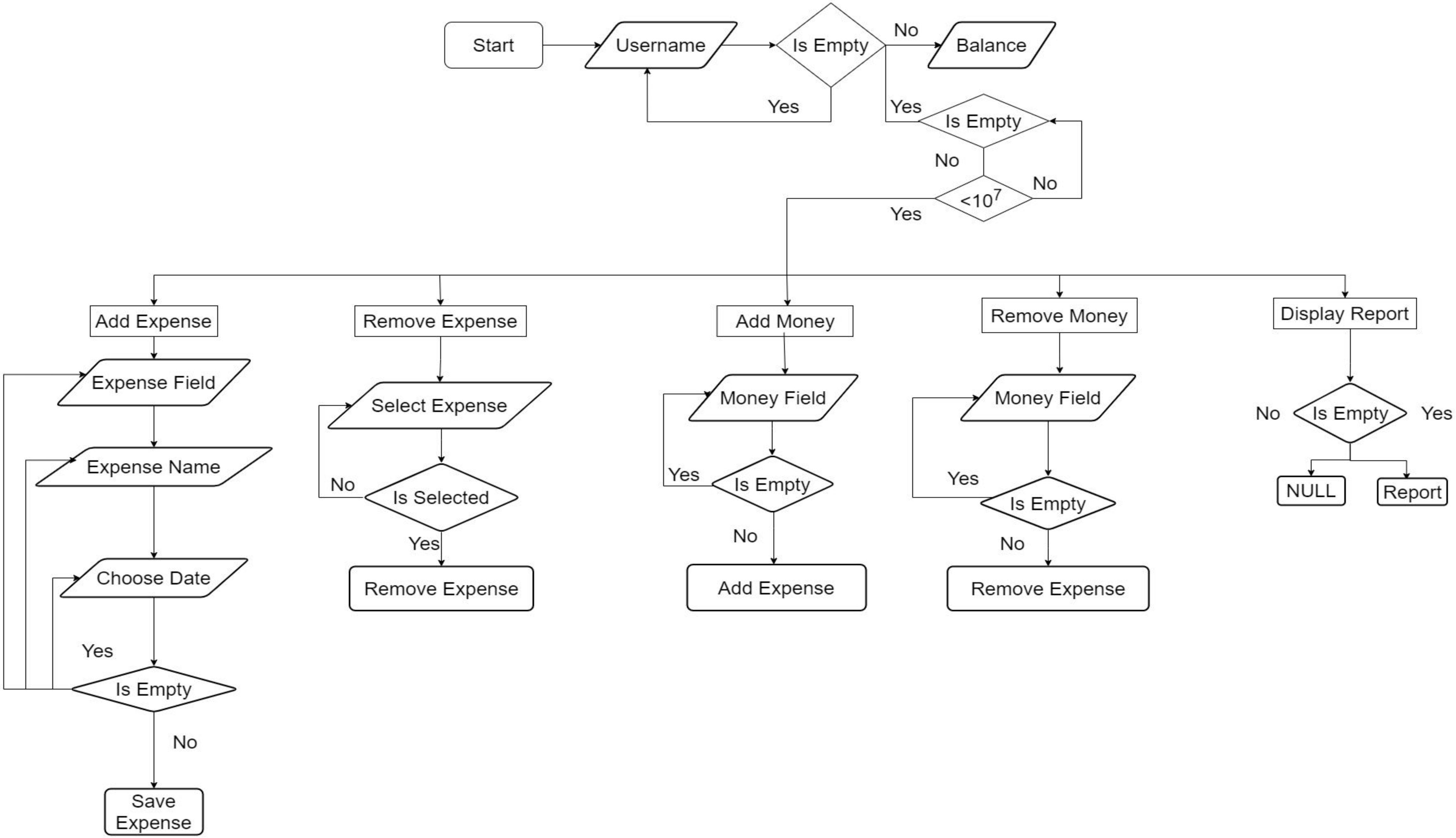
⭐Control Flow Diagram:

- A **control-flow diagram** (**CFD**) is a diagram to describe the control flow of a business process, process or review.

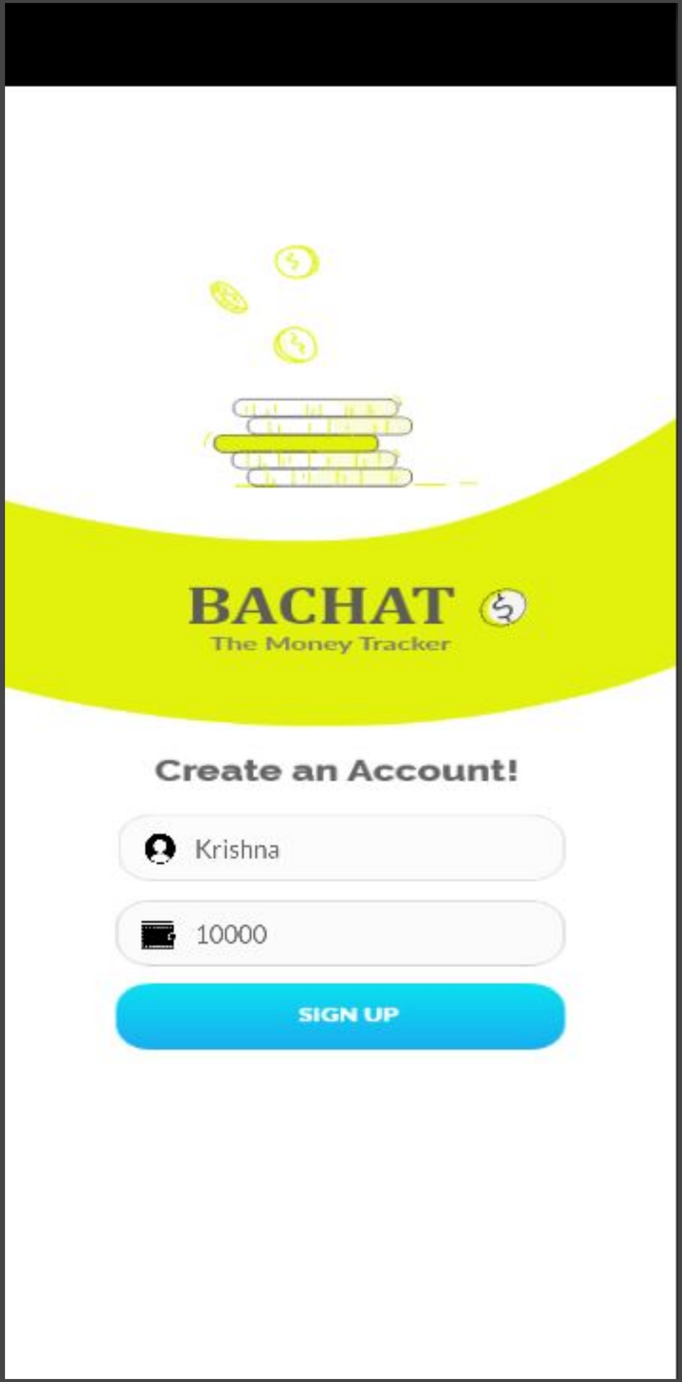
- They are one of the classic business process modeling methodologies, along with flow charts, drakon-charts, data flow diagrams, functional flow block diagram, Gantt charts, PERT diagrams, and IDEF.

- A control-flow diagram can consist of a subdivision to show sequential steps, with if-then-else conditions, repetition, and/or case conditions. Suitably annotated geometrical figures are used to represent operations, data, or equipment, and arrows are used to indicate the sequential flow from one to another

Control Flow Diagram



Application Overview



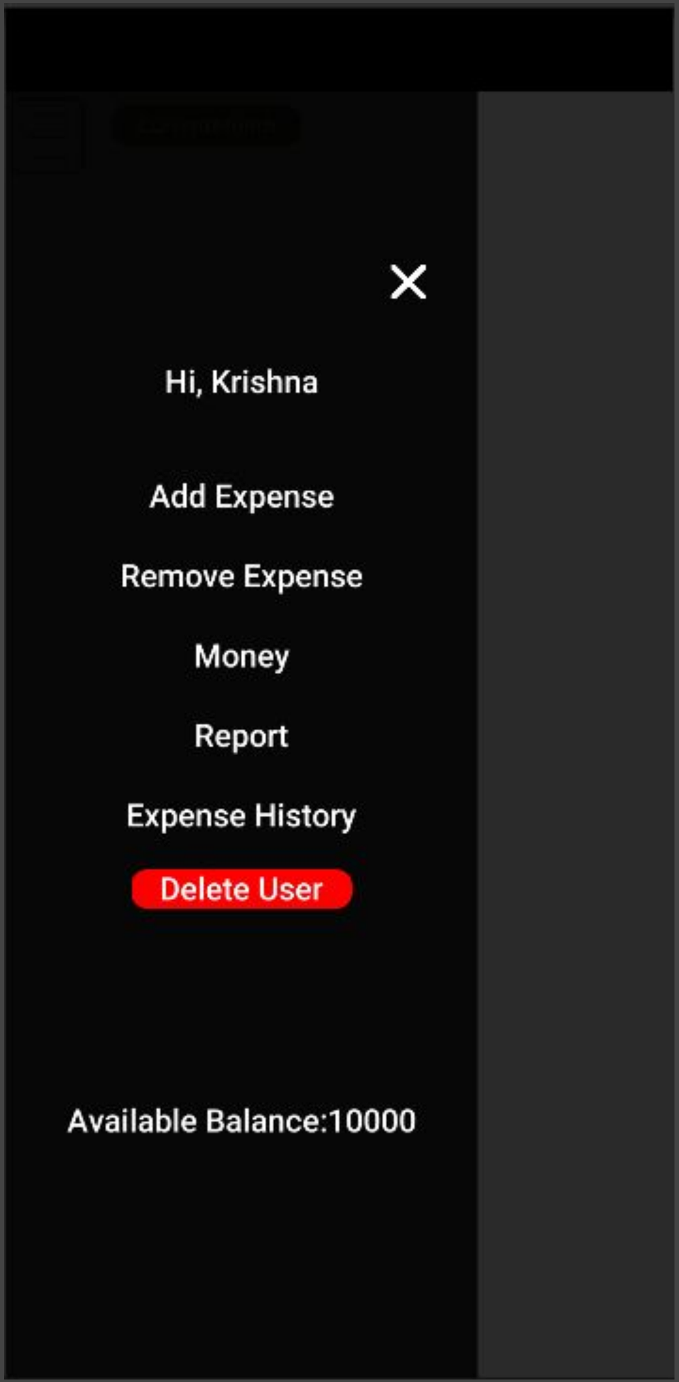
BACHAT

The Money Tracker

Create an Account!

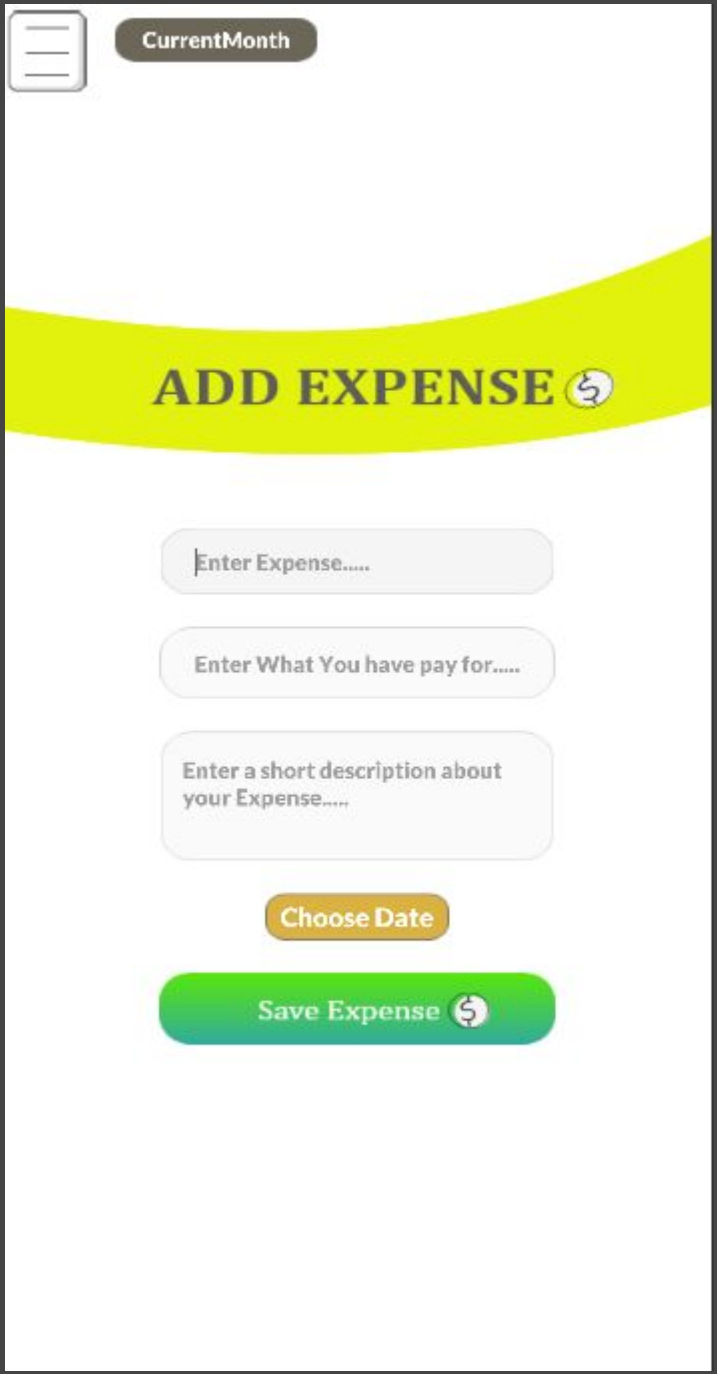
Krishna

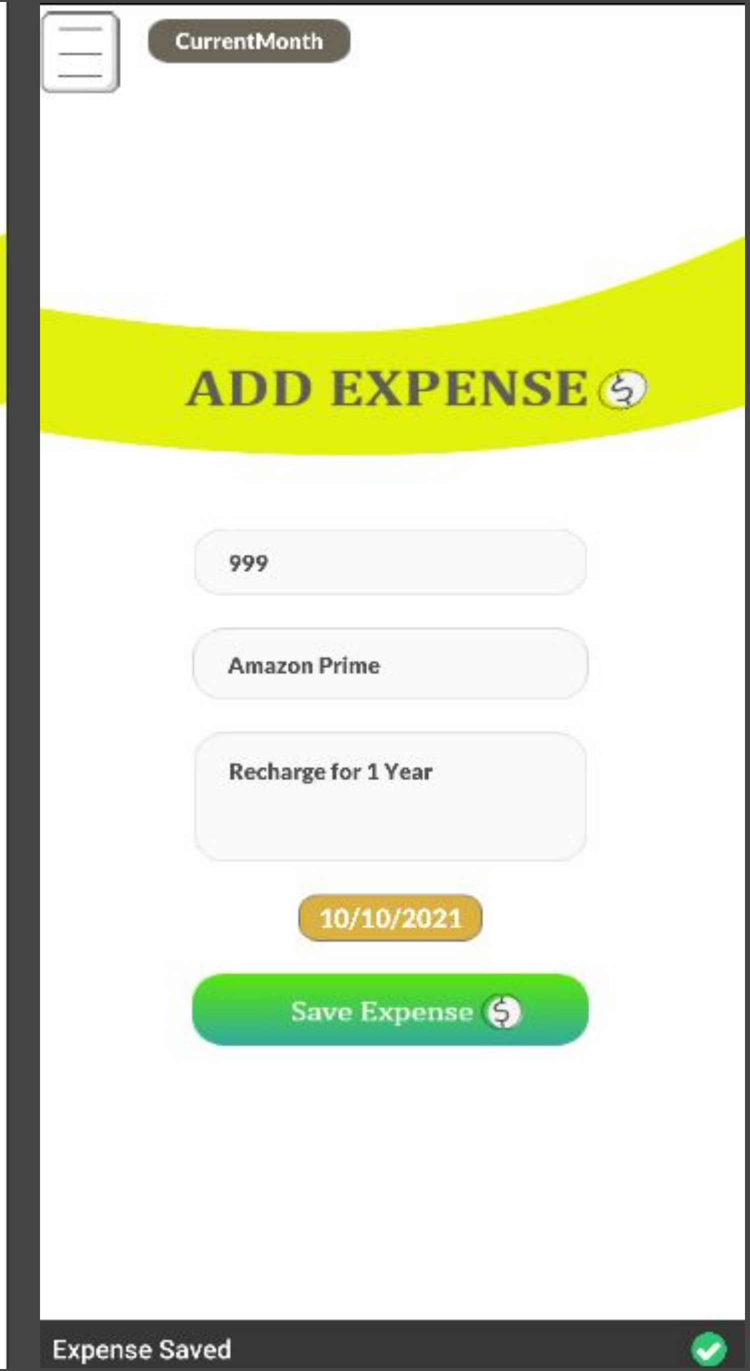
10000



Hi, Krishna

Available Balance:10000


CurrentMonth
ADD EXPENSE
Enter Expense.....
Enter What You have pay for.....
Enter a short description about your Expense.....
Choose Date
Save Expense



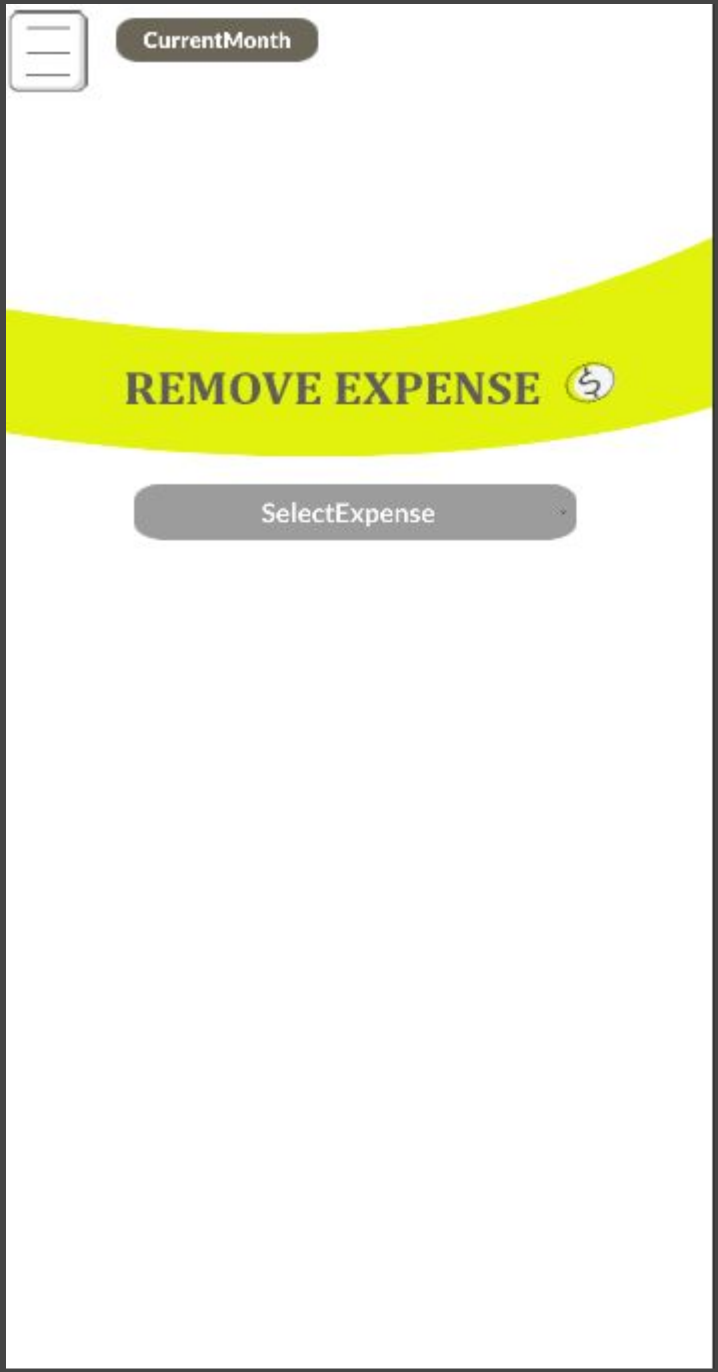
999

Amazon Prime

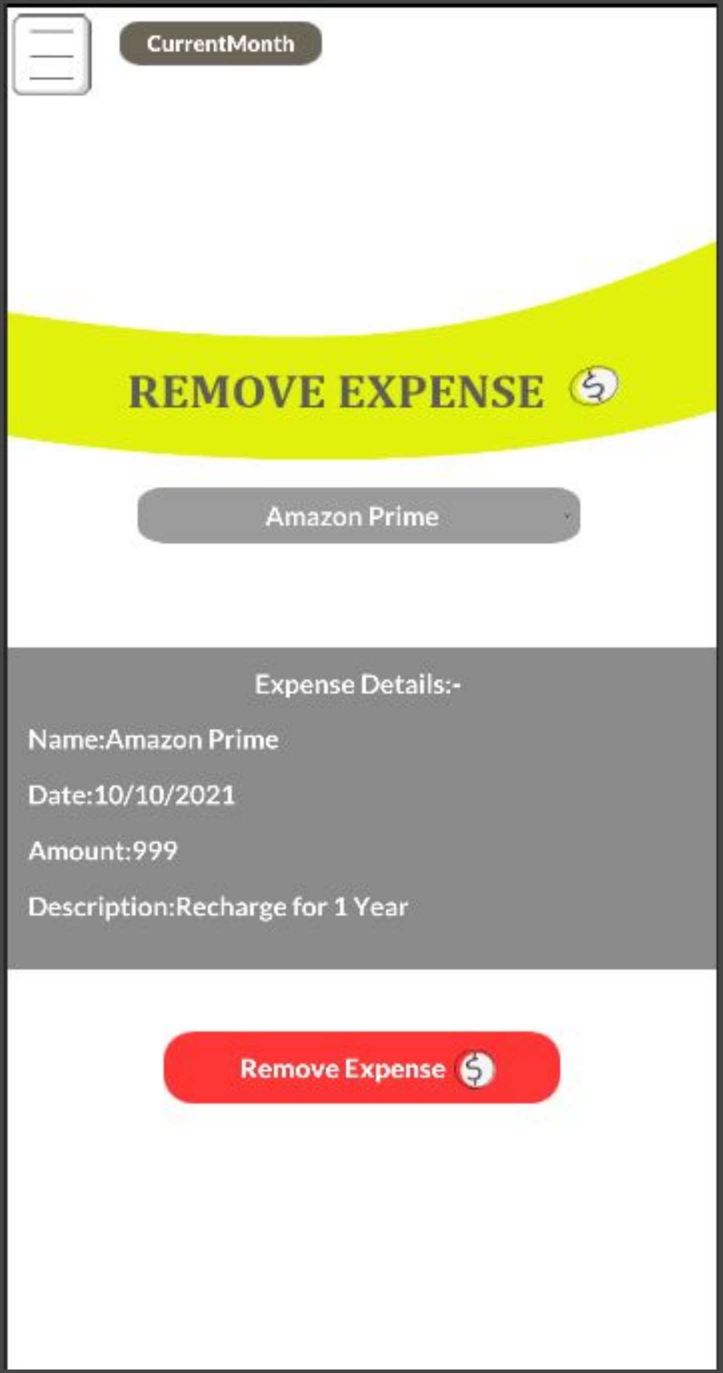
Recharge for 1 Year

10/10/2021

Expense Saved ✓



REMOVE EXPENSE



REMOVE EXPENSE

Amazon Prime

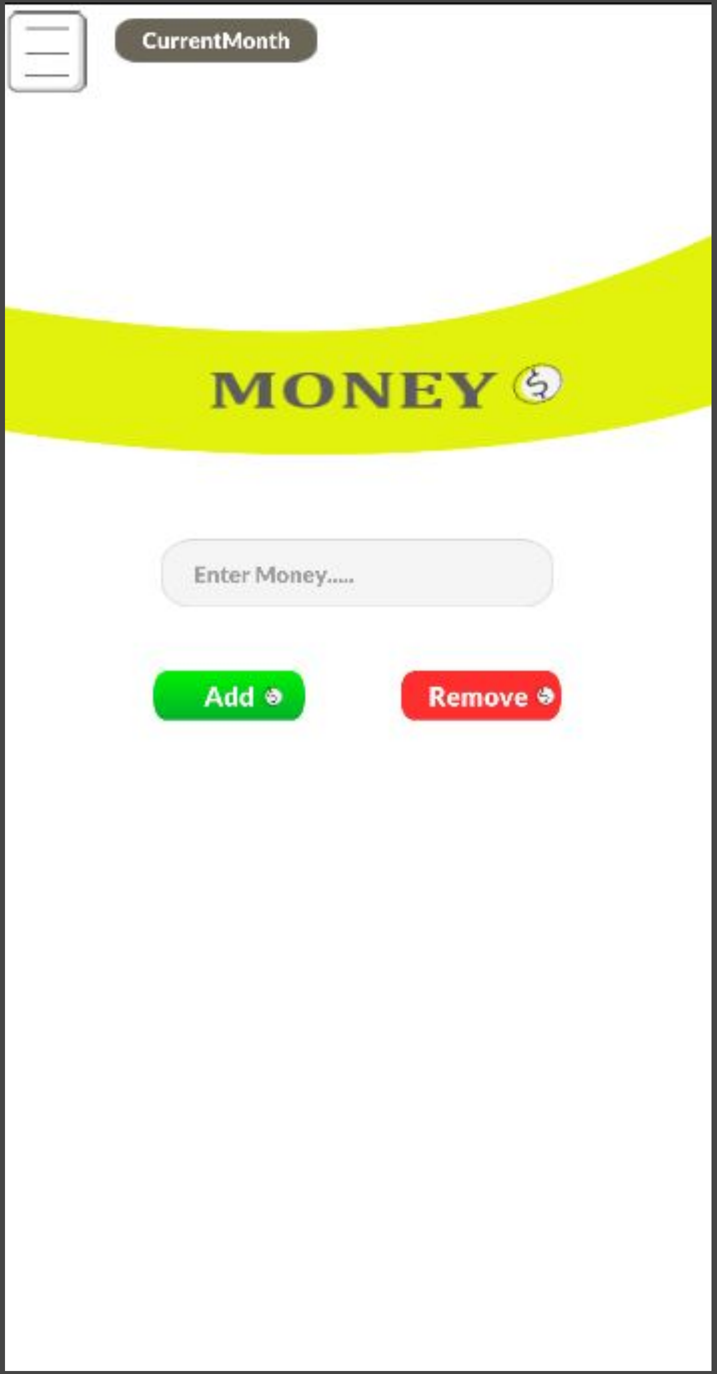
Expense Details:-

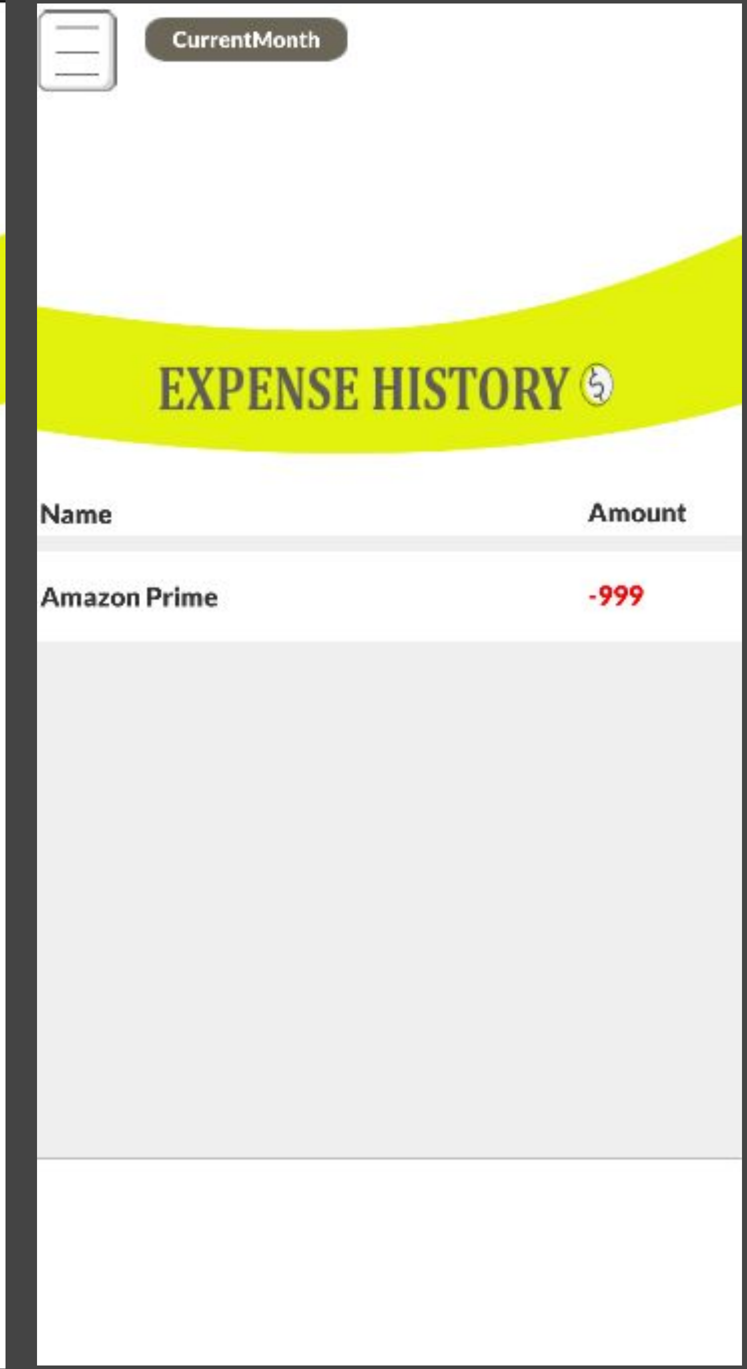
Name:Amazon Prime

Date:10/10/2021

Amount:999

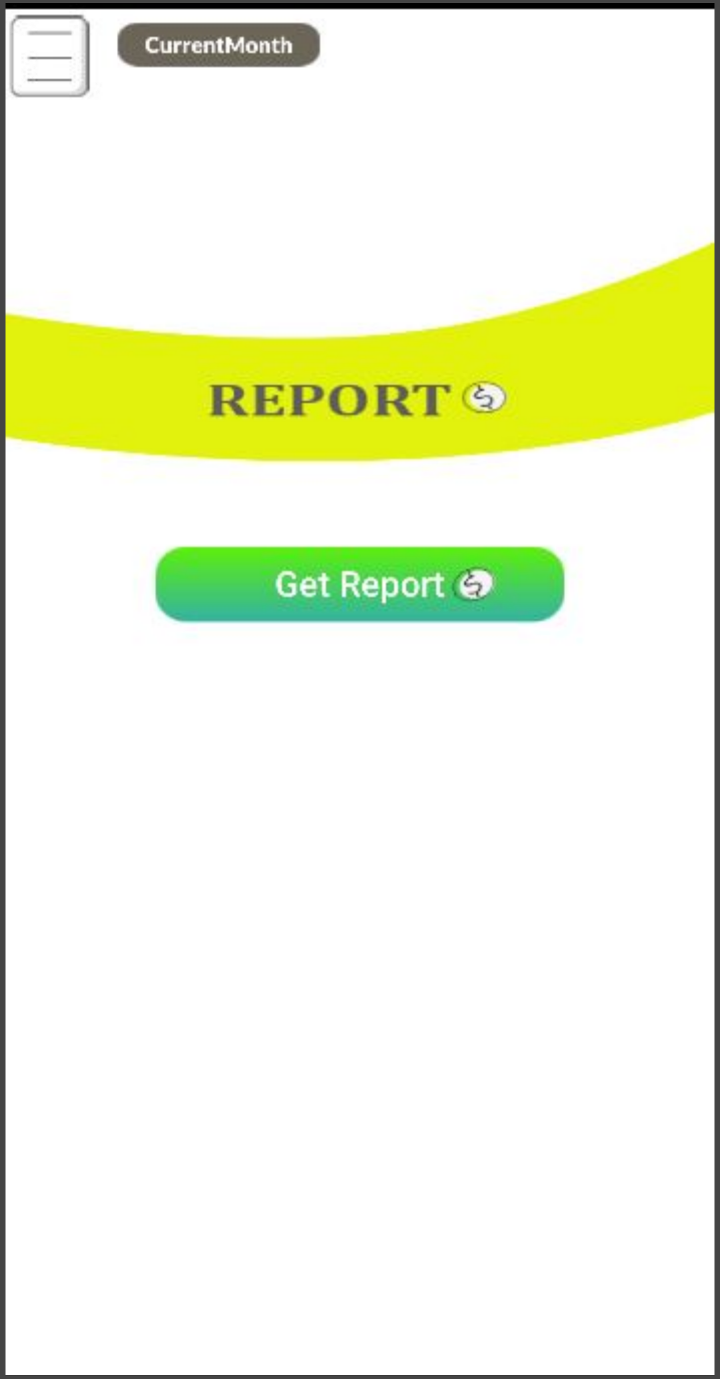
Description:Recharge for 1 Year

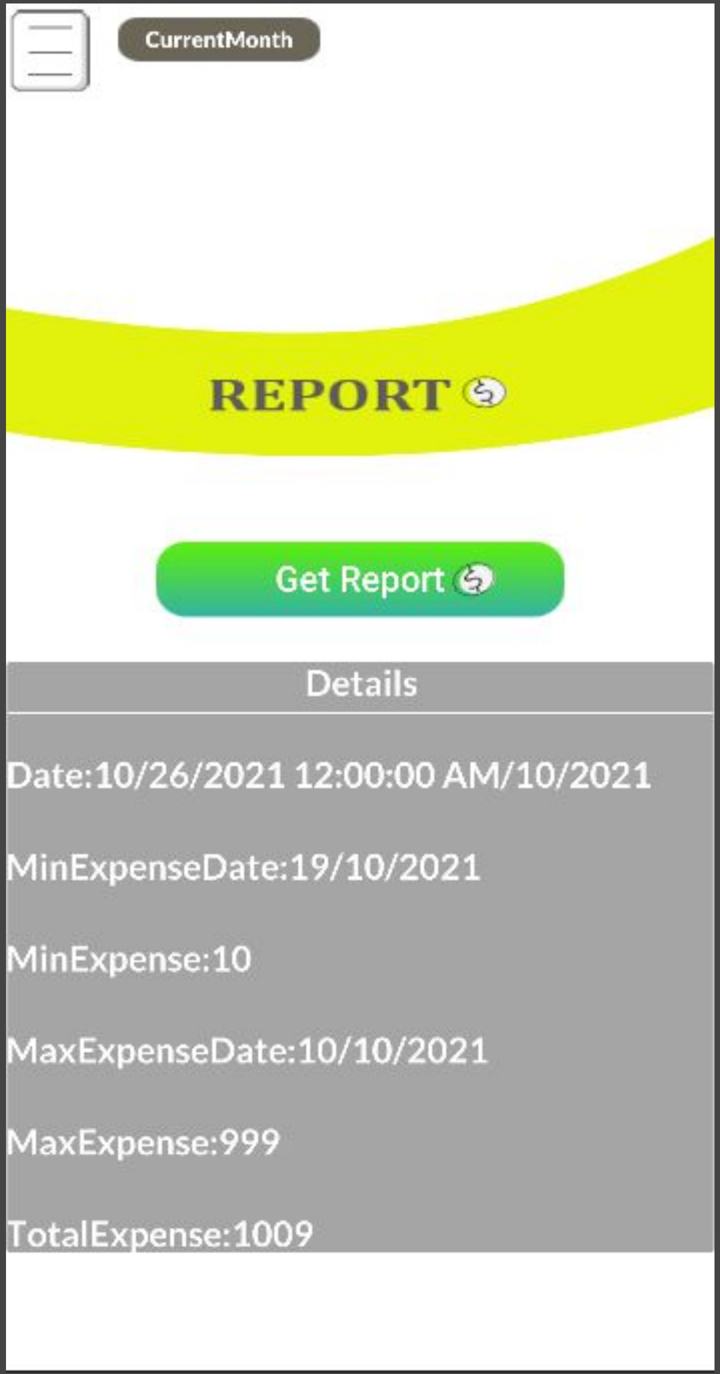




EXPENSE HISTORY

Name	Amount
Amazon Prime	-999





Details

Date:10/26/2021 12:00:00 AM/10/2021

MinExpenseDate:19/10/2021

MinExpense:10

MaxExpenseDate:10/10/2021

MaxExpense:999

TotalExpense:1009

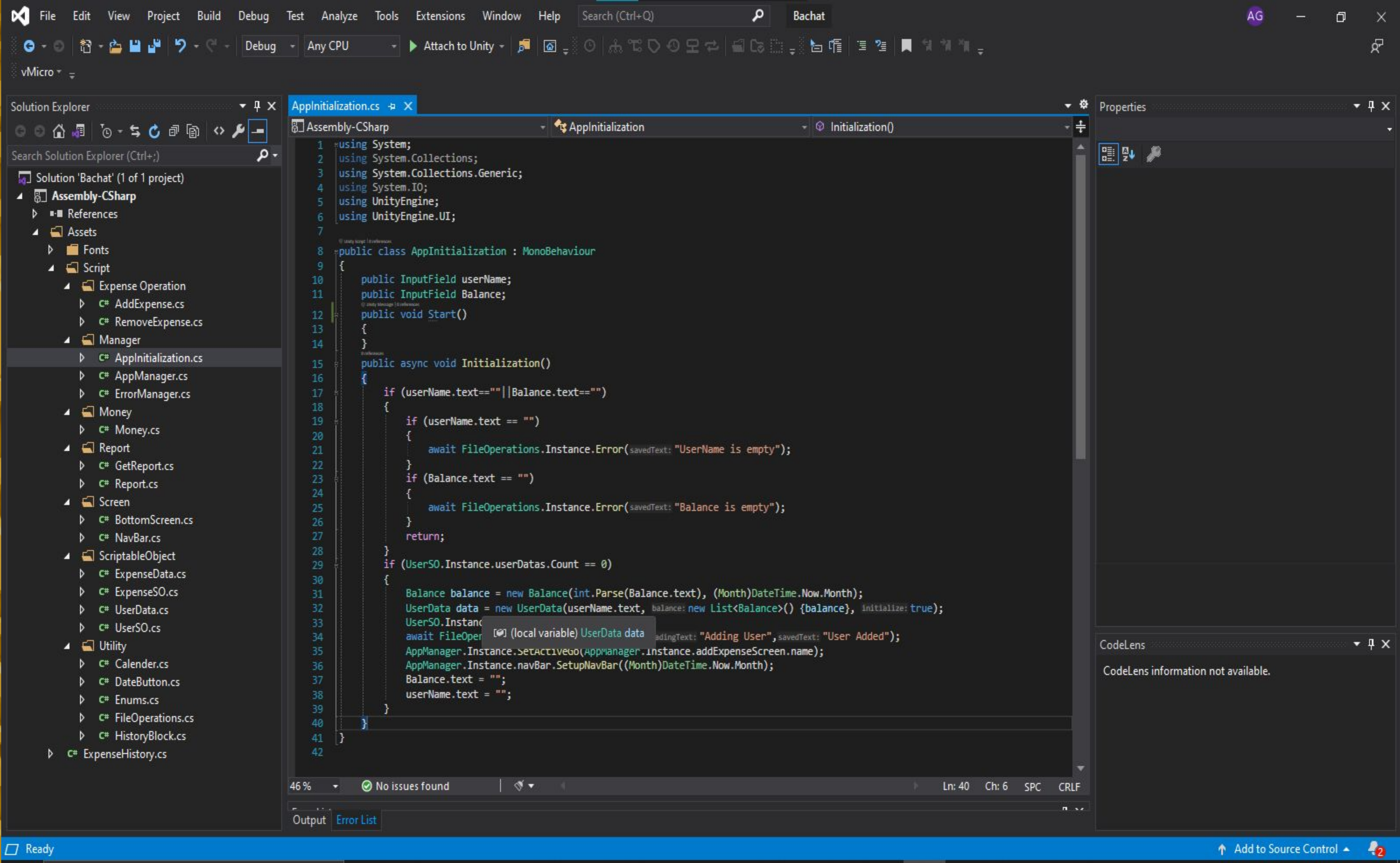
App Demo

Click here to watch Demo

Note:- Link will be clickable in presentation mode

Code

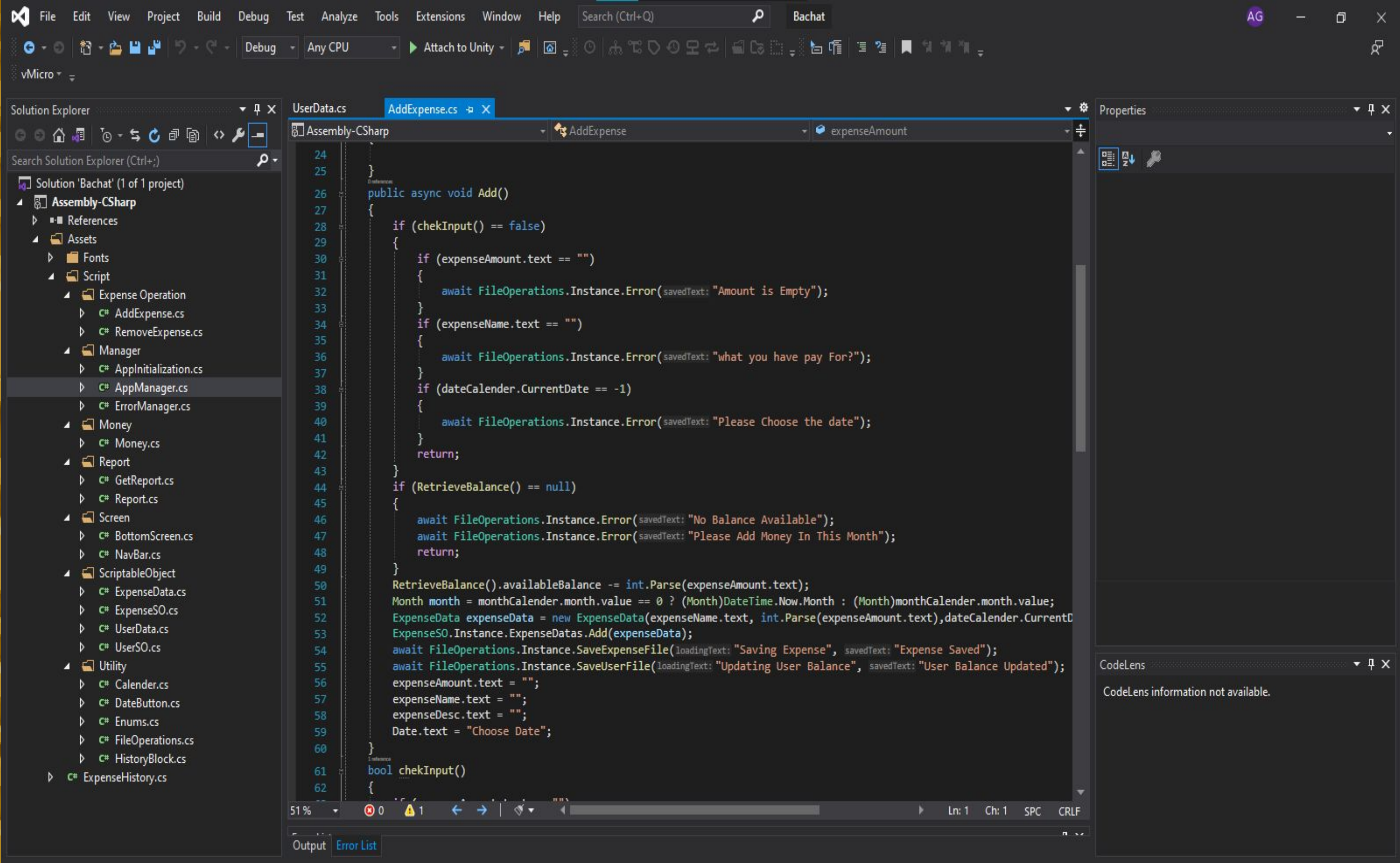
Screenshots



```csharp
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using UnityEngine;
using UnityEngine.UI;

public class AppInitialization : MonoBehaviour
{
    public InputField userName;
    public InputField Balance;
    public void Start()
    {
    }
    public async void Initialization()
    {
        if (userName.text==""||Balance.text=="")
        {
            if (userName.text == "")
            {
                await FileOperations.Instance.Error(savedText: "UserName is empty");
            }
            if (Balance.text == "")
            {
                await FileOperations.Instance.Error(savedText: "Balance is empty");
            }
            return;
        }
        if (UserSO.Instance.userDatas.Count == 0)
        {
            Balance balance = new Balance(int.Parse(Balance.text), (Month)DateTime.Now.Month);
            UserData data = new UserData(userName.text, balance: new List<Balance>() {balance}, initialize: true);
            UserSO.Instanc
            await FileOper                                    adingText: "Adding User",savedText: "User Added");
            AppManager.Instance.SetActiveGo(AppManager.Instance.addExpenseScreen.name);
            AppManager.Instance.navBar.SetupNavBar((Month)DateTime.Now.Month);
            Balance.text = "";
            userName.text = "";
        }
    }
}
```

(local variable) UserData data



```
    }
    public async void Add()
    {
        if (chekInput() == false)
        {
            if (expenseAmount.text == "")
            {
                await FileOperations.Instance.Error(savedText: "Amount is Empty");
            }
            if (expenseName.text == "")
            {
                await FileOperations.Instance.Error(savedText: "what you have pay For?");
            }
            if (dateCalender.CurrentDate == -1)
            {
                await FileOperations.Instance.Error(savedText: "Please Choose the date");
            }
            return;
        }
        if (RetrieveBalance() == null)
        {
            await FileOperations.Instance.Error(savedText: "No Balance Available");
            await FileOperations.Instance.Error(savedText: "Please Add Money In This Month");
            return;
        }
        RetrieveBalance().availableBalance -= int.Parse(expenseAmount.text);
        Month month = monthCalender.month.value == 0 ? (Month)DateTime.Now.Month : (Month)monthCalender.month.value;
        ExpenseData expenseData = new ExpenseData(expenseName.text, int.Parse(expenseAmount.text),dateCalender.CurrentD
        ExpenseSO.Instance.ExpenseDatas.Add(expenseData);
        await FileOperations.Instance.SaveExpenseFile(loadingText: "Saving Expense", savedText: "Expense Saved");
        await FileOperations.Instance.SaveUserFile(loadingText: "Updating User Balance", savedText: "User Balance Updated");
        expenseAmount.text = "";
        expenseName.text = "";
        expenseDesc.text = "";
        Date.text = "Choose Date";
    }
    bool chekInput()
    {
```

⭐Limitations:

- Multiple users are not supported by this app.
- There is no database in this app.
- User data is not encrypted in any way.

⭐Future Scope:

- We all know that the majority of transactions are done online, thus by using this app, we can automatically store all of our online transactions, allowing this software to analyse your spending track in greater detail.

- This will replace the use of a bank passbook, reducing the use of paper, and eliminating the requirement for a person to go to the machine to print the passbook, saving both time and money.

Thank You

Submitted By:

EN19CS305014	Atharva Gangrade
EN19CS305032	Nimish Porwal
EN19CS305037	Sagar Yadav