

# Advanced Prototype

## BudgetWidget



### OVERVIEW

Welcome back to our Third face of the development, the Advanced prototype. We here present to you our next phase in our application development after our successful completion of the Basic Prototype.

In this phase, our requirements have been enhanced and the given set of requirements also includes some alterations to be done over the previously implemented prototype. We have adapted ourselves to these dynamic requirements and modified the code accordingly.



### Additional Requirements of the Project

#### Andriod Application Developement (*Advanced Prototype*)

Essentials	Necessary	Nice to Have
Different categories	Icons	Currencies +plus conversion
Add Date.Month.Year	Delete entries	Help menu
Filters ( <i>by category, Month, etc</i> )	Budget threshold	<b>Analytics</b>
Additional notes per entry	Repeated transactions	<b>Show costs in chart</b>
Mode of payment	<b>Log in with ID and password</b>	
<b>Easy to enter numbers</b>	<b>Start screen indicating App purpose</b>	
<b>Select the currency in settings</b>	<b>Access to contacts (e.g., if money is lent to a friend)</b>	



## Software Architecture

The architecture of a system describes its major components, their relationships (*structures*), and how they interact with each other. Architecture serves as a blueprint for a system. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components. Therefore, it plays a significant role and helps in minimizing expenses. It adds structure procedure in development, reliable foundation, well-defined points for extensions, and no uncontrolled “*balcony additions*”.

The architecture also is the primary carrier of system qualities such as performance, modifiability, and security, none of which can be achieved without a unifying architectural vision.

Design Pattern is of three main classifications.

◆  
Creational pattern  
Structural pattern  
Behavioral pattern  
◆

We have decided to use two design patterns throughout our development.

They are the **Structural: Composite Design Pattern** and  
**Model-View-Controller**.

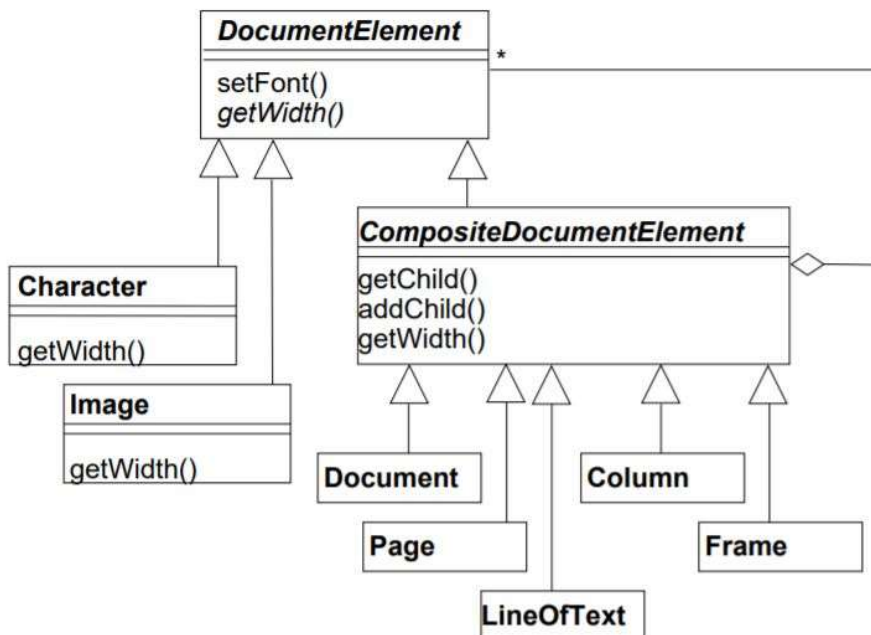


### Structural Pattern: Composite

A composite pattern is a partitioning design pattern and describes a group of objects that are treated the same way as a single instance of the same type of object. The intent of a composite is to “compose” objects into tree structures to represent part-whole hierarchies. In other words, it consists of an abstract interface for “leaves” branching and nodes in a tree

Applications include

Flexible access layer for file system  
Part structure for devices  
Genealogical tables(trees)



### Model View Controller

MVC is short for Model, View, and Controller. MVC is a popular way of organizing your code. The big idea behind MVC is that each section of your code has a purpose, and those purposes are different. Some of your code holds the data of your app, some of your code makes your app look nice, and some of your code controls how your app

functions. MVC is a way to organize your code's core functions into their own, neatly organized boxes. This makes thinking about your app, revisiting your app, and sharing your app with others much easier and cleaner.

### Model

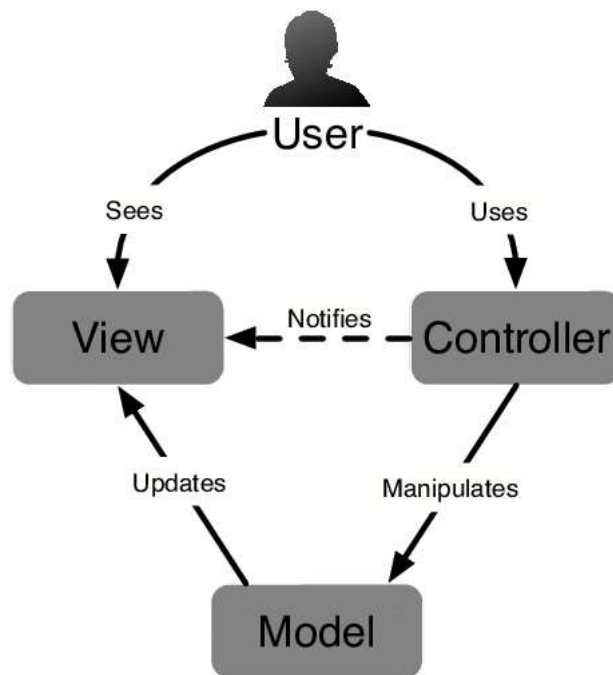
Model code typically reflects real-world things. This code can hold raw data, or it will define the essential components of your app. For instance, if you were building a To-do app, the model code would define what a "task" is and what a "list" is – since those are the main components of a to-do app.

### View

View code is made up of all the functions that directly interact with the user. This is the code that makes your app look nice, and otherwise defines how your user sees and interacts with it.

### Controller

Controller code acts as a liaison between the Model and the View, receiving user input and deciding what to do with it. It's the brains of the application, and ties together the model and the view



### Why MVC?

Faster Development Process

Ability To Provide Multiple Views

Support For Asynchronous Technique

The Modification Does Not Affect The Entire Model

MVC Model Returns The Data Without Formatting



## Coding Conventions

Coding conventions are a set of guidelines for a specific programming language that recommend programming style, practices, and methods for each aspect of a program written in that language. These conventions usually cover file organization, indentation, comments, declarations, statements, white space, naming conventions,

The two most popular, IntelliJ IDEA and Android Studio provide powerful support for code styling. You can configure them to automatically format your code inconsistency with the given code style.

### Naming Conventions

Naming conventions make programs more understandable by making them easier to read. They can also give information about the function of the identifier-for example, whether it's a constant, package, or class-which can be helpful in understanding the code.

### CamelCase

CamelCase is a naming convention in which the first letter of each word in a compound word is capitalized, except for the first word. Software developers often use camelCase when writing source code. CamelCase is useful in programming since element names cannot contain spaces. The CamelCase naming convention makes compound names more readable. For example, myOneMethod is easier to read than myonemethod.

Combining Camel Case with lowercase

### **lowerCamelCase**

lowerCamelCase (part of CamelCase) is a naming convention in which a name is formed of multiple words that are joined together as a single word with the first letter of each of the multiple words (except the first one) capitalized within the new word that forms the name which is opposite to UpperCamelCase.

```
switch(data.getItem()){
    case "Transport":
        holder.imageView.setImageResource(R.drawable.ic_tr
        break;
    case "Food":
        holder.imageView.setImageResource(R.drawable.ic_fo
        break;
    case "House":
        holder.imageView.setImageResource(R.drawable.ic_ho
        break;
    case "Entertainment":
        holder.imageView.setImageResource(R.drawable.ic_en
        break;
    case "Education":
        holder.imageView.setImageResource(R.drawable.ic_ed
        break;
    case "Charity":
        holder.imageView.setImageResource(R.drawable.ic_co
        break;
    case "Apparel":
        holder.imageView.setImageResource(R.drawable.ic_sh
        break;
    case "Health":
        holder.imageView.setImageResource(R.drawable.ic_he
        break;
    case "Personal":
        holder.imageView.setImageResource(R.drawable.ic_pe
        break;
    default:
        holder.imageView.setImageResource(R.drawable.ic_ot
        break;
}
```

> Preview of a part of our code where we have used lowerCamelCase

### **Snake\_case**

snake\_case is a naming convention in which a developer replaces spaces between words with an underscore. Most object-oriented programming languages don't allow variable, method, class, and function names to contain spaces.

The snake case -- also commonly seen as snake\_case - naming convention replaces spaces with underscores to create descriptive variable and method names from multiple words without a compile-time error

```

//textviews
progress_ratio_transport = findViewById(R.id.progress_ratio_transport)
progress_ratio_food = findViewById(R.id.progress_ratio_food)
progress_ratio_house = findViewById(R.id.progress_ratio_house)
progress_ratio_ent = findViewById(R.id.progress_ratio_ent)
progress_ratio_edu = findViewById(R.id.progress_ratio_edu)
progress_ratio_cha = findViewById(R.id.progress_ratio_cha)
progress_ratio_app = findViewById(R.id.progress_ratio_app)
progress_ratio_heal = findViewById(R.id.progress_ratio_heal)
progress_ratio_per = findViewById(R.id.progress_ratio_per)
progress_ratio_oth = findViewById(R.id.progress_ratio_oth)

//imageviews
status_image_transport = findViewById(R.id.status_image_transport)
status_image_food = findViewById(R.id.status_image_food)
status_image_house = findViewById(R.id.status_image_house)
status_image_ent = findViewById(R.id.status_image_ent)
status_image_edu = findViewById(R.id.status_image_edu)
status_image_cha = findViewById(R.id.status_image_cha)
status_image_app = findViewById(R.id.status_image_app)
status_image_heal = findViewById(R.id.status_image_heal)
status_image_per = findViewById(R.id.status_image_per)
status_image_oth = findViewById(R.id.status_image_oth)

```

> Preview of a part of our code where we have also used snake\_case



## Context of Use

The Context of Use is the actual conditions under which a given artifact/software product is used, or will be used in a normal day-to-day working situation. It is important to carry out usability tests, prototyping sessions, meetings, user studies, and other "user-dependent sessions" in the context of use to get as high ecological validity (see this) of your findings as possible

The data for a context of use analysis can be gathered using interviews, workshops, surveys, site visits, focus groups, observational studies, and so on. Collecting and analyzing detailed information about the intended users, their tasks, and the technical and environmental constraints is certainly of primary importance.


### Goals

- Ensure that all factors that relate to the use of the system are identified before design work starts.
- Provide a basis for designing later usability tests.

We have to build 2 personal profiles to describe the idea of the users who might be using our Android Application. In this way, we would be able to observe and understand the individuals' motivation, personality, patterns, goals, and tasks within our app domain.

**Student: Johnny Sins**

**Traveling salesWomen: Roxanne Fernandes**



## Johnny Sins


Student

**Demography**  
**19 Years**

- Male
- Student at Otto-von-Guericke-Universität.

**Goals**

- Organize my expense.
- Recurring transaction must be added without me having to manually add every time.
- I need a visual/ graphical representation of my expenditure




**Background and Drawbacks**


Being an Student and managing your expense is very difficult and therefore even more essential.

I lose track of my daily expenses and towards the end of the month have overlap, therefore, I want to manage my expense within my fixed budget so I don't have to work unplanned part time jobs and lose touch of studies.

**Graphical View**



Here is one of our - might be potential user Johnny, who is not just having trouble organizing his expense as a student but also wants a graphical view of his expense for a quick and enhanced overview.



## Roxanne Fernandes

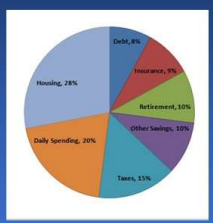
Employer

**Demography**  
**39 Years**

- Female
- CEO and Founder of Facial Cosmetics

**Goals**

- Day to day expense must monitored
- Set Monthly Budget/ Threshold
- Track Monthly Expense Get alert in case of overlap of Budget.




**Background and Drawbacks**

Being an Employer it is essential to monitor your expenses on daily basis and often get lose of my expense as I travel frequently due to business.

I'm unable to check daily expense and Maintain monthly budget threshold and distinct business expenses from personal expense for tax purpose is very important.

**Alert on Threshold**



Our second possible user could be someone like Roxanne who travels frequently due to business and wants to keep track of monthly personal expenses by setting a budget threshold and wants an alert ( flags ) to show her expenses against the set budget.



## Design Solution

Our design solution is to provide the user with all the essential and required functionalities with a simple and basic user interface

When we planned the design, we wanted the most user-friendly interface. We designed the interface considering two things:

- Familiar and Intuitive interface.
- Simple and easy to access interface.

We used the following features efficiently to attain our goal:

### GRID layout

The Grid layout incorporates all functionalities of the App put up as a view on the main page for users to access easily.

### Security

The user security is provided with a mandatory field where the user needs to enter their credentials to access their personal expense account.



### Floating button

A floating button is used for log out and User information to make it more convenient for our user and in addition to that a pop-up text saying "are you sure?" is added to minimize unintentional logout.

### Help tab

Gives you the threshold and amounts related to savings irrespective of any item or category.

### Analytics

Gives user visual/graphical representation of the expense along with 3 flags indicating the measurement of the expense in relation to the budget set.

### Number Keypad

Gives the user an automatically Number keypad in place of amount instead of a qwerty keypad.

### Currency Spinner

Gives user few currency options which can be chosen, like Euros, Dollar, Pound, INR from

### Offline Capabilities

Users can use the App without the internet and the data will be stored internally in the cash memory of firebase and updated to firebase whenever the user connects to the internet. This ensures user-friendly and hassle-free.

The below figures give a visual representation of some of our design solutions.

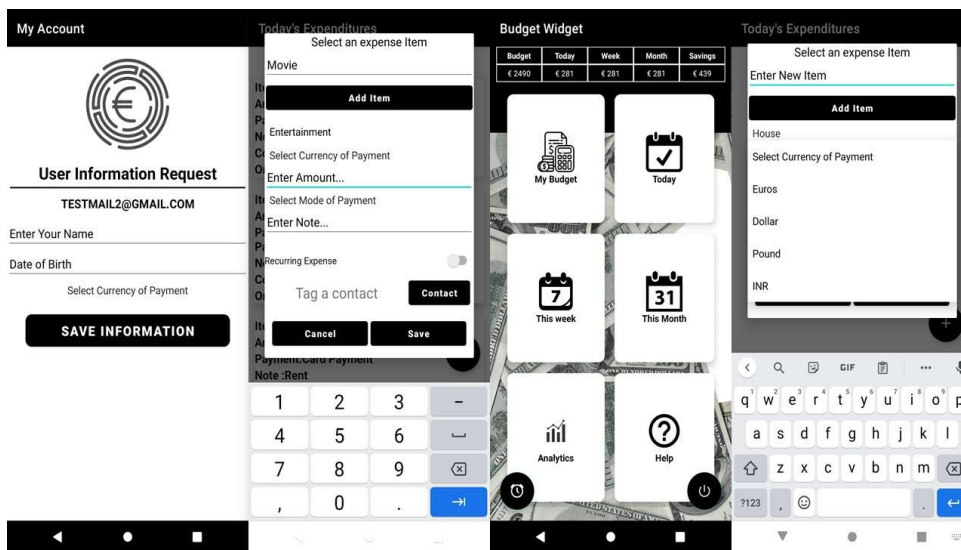


Fig 1: Security, Fig 2: Number Keypad, Fig 3: GRID layout, Fig 4: Currency Spinner.



## Summary of changes

The following is the overview of the changes and additions to our basic prototype App.

#### • Transaction page:

(before) qwerty keyboard.

(latest) Number keypad;

#### • Setting page:

(before) No filters.

(latest) select Currency: default currency;

#### • Threshold tab:

(before) No filter.

(latest) Filters based on Category. Monthly, Weekly;

#### • Login:

(before) Simple login.

(latest) Login with password & username has been added and stores in database - mandatory;

#### • Splash screen:

(before) No display of App purpose.

(latest) display as "App that controls your money";

• **Transaction page:**

**(before)** Manually input transaction ID.

**(latest)** Access to contact to take ID from the contacts stored in the app;

• **Database:**

**(before)** firebase: app required internet to run.

**(latest)** The app can fully run without the internet also;

• **Analytics page: ( New )**

Today/ Weekly/ Monthly threshold analytical graph implement for a user to view their expenses and incomes along with a pie chart and an alert which indicates as follows

- ♦ Less than 50% of the budget used: **Green flag** ✓
- ♦ Between 50% to 100% of the budget used: **Brown flag**
- ♦ More than 100% of the budget used: **Red flag** 🚩

• **User stories:**

The new essential requirement that we had to add to the user story was to enable users to add a category, the balance sheet on HOME UI top of the main menu, currency conversion. Manipulate transaction entries and change currency.



USER STORIES
As a User, I want to sign up to log in to the app.
As a User, I want to number pad to enter numbers in currency place.
As a User, I want to select default currency in settings.
As a User, I want to view what the app is used for.
As a User, I want to store my login credentials to be accessed it anytime.
As a User, I want to access my contacts and use them in my transaction ID.
As a User, I want to view the budget threshold.
As a User, I want to have icons for currency (currency symbol).
As a User, I want to view the analytics of my expense.
As a User, I want to alert me with different flags when I cross the budget threshold.
As a User, I want to run the app and use it in offline mode.

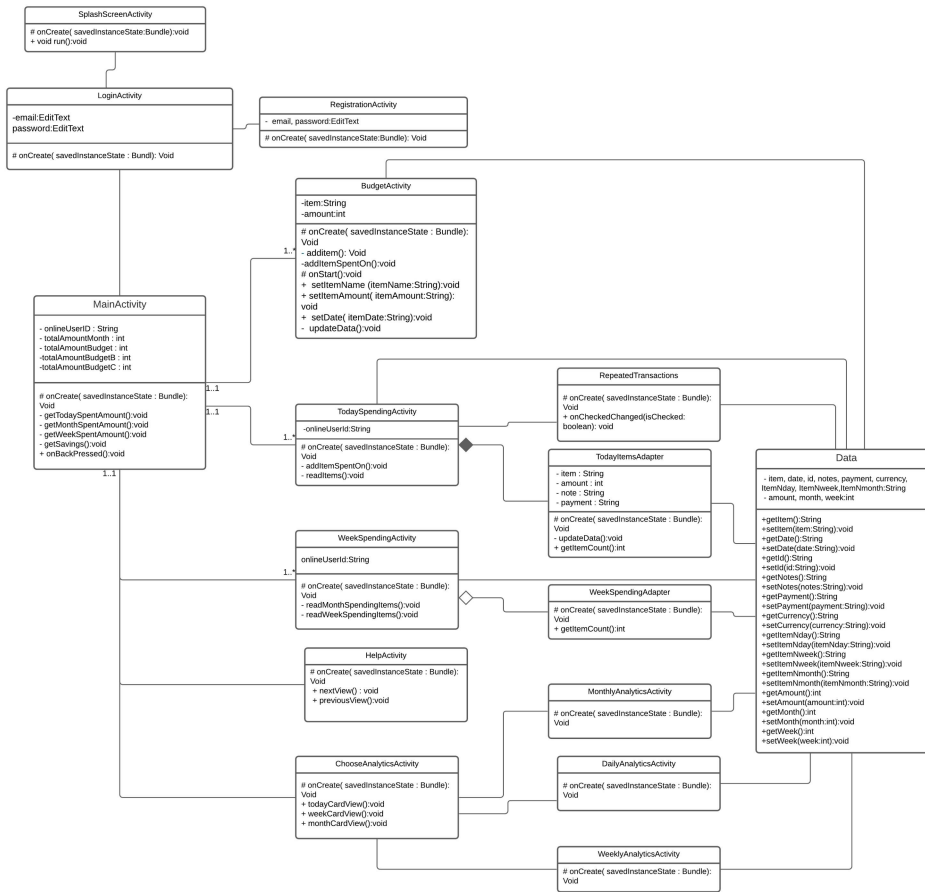


## Class Diagram

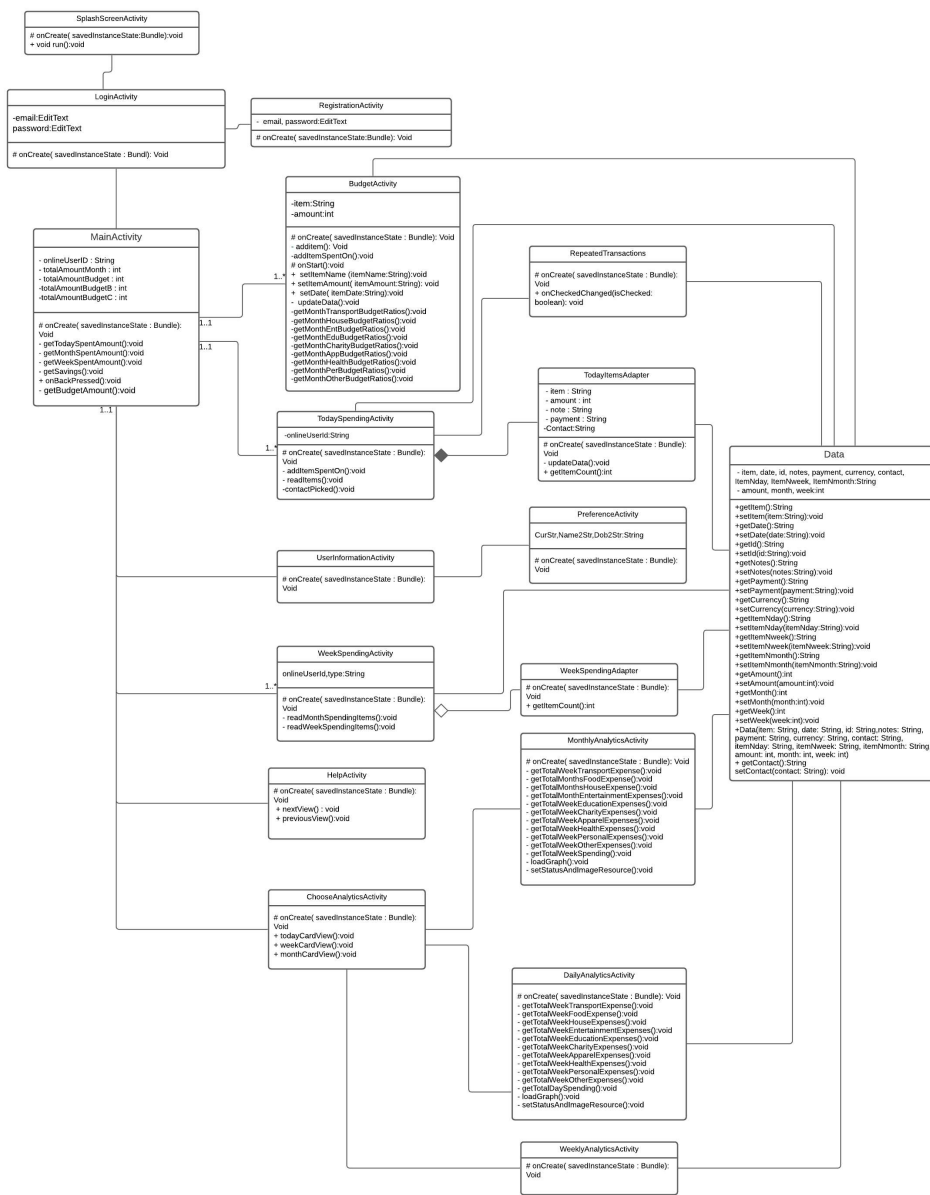
Changes to Main Activity, Budget Activity, Today's spending Activity, Monthly/ Daily / Weekly Analytics Activity, and Data Implemented along with two new classes User Information Activity and Preference Activity.



## Our Previous Basic Prototype class diagram

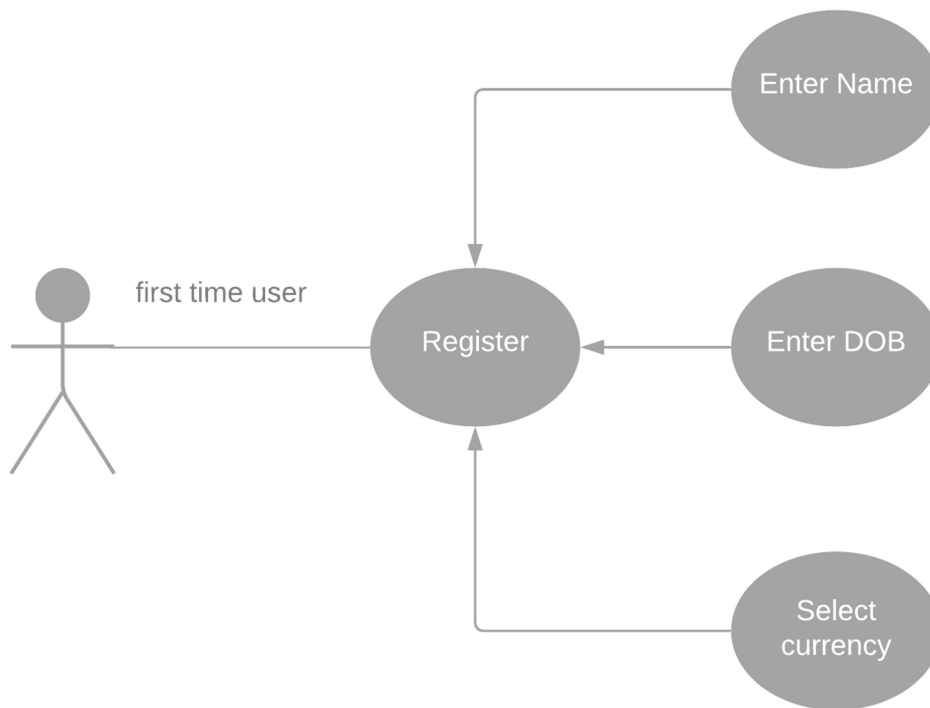


## Our Updated Class Diagram

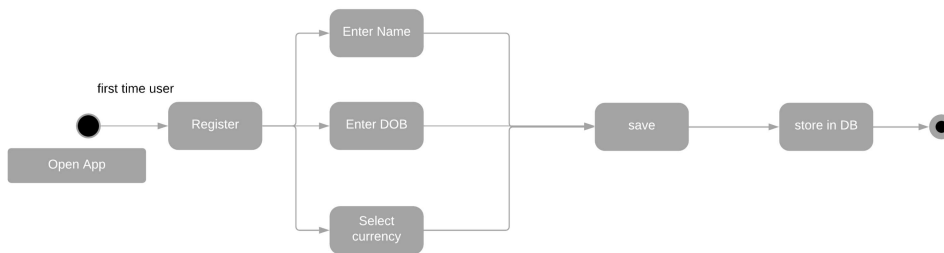


## Use-Case Diagram

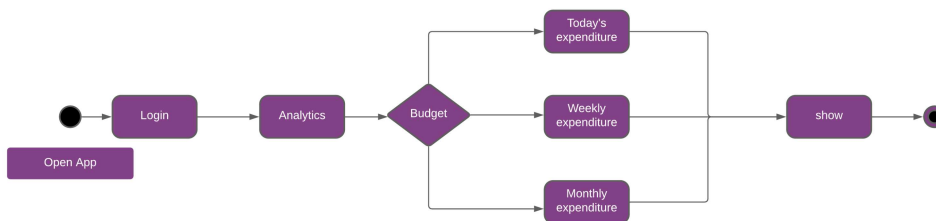
Below you can view our updated and Additional Use-Case Diagrams for our Advanced prototype features.



The above Use-case diagram shows the basic login functionality for registering for a new user where the user needs to enter Name, Date of birth, and select default currency.



The above Use-case diagram shows the data saved in the database once the user has finished the registration.



The above Use-case diagram shows that the user can select view Transaction Analytics against the Budget threshold by "Todays Expenditure", "Weekly expenditure ", and " Monthly expenditure "



## Advance Working prototype

Our final content, App preview/ Advanced Working prototype to show an overview of the app experience, focusing on the app's additional features to basic prototype to enhance better user experience.

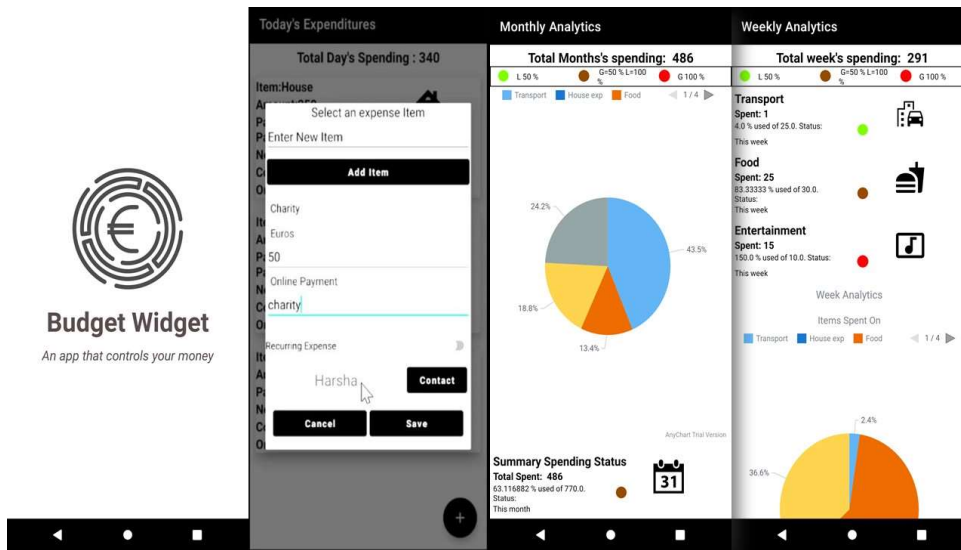


Fig 1, illustrates the Splash Screen activity, quoting "An app that controls your money"

Fig 2, You can see that Users can access their contacts and take the ID from their contact list in any given transactions without having the hassle to manually input it every time.

Fig 3, 4: shows the graphical representation of Monthly and weekly analytics, with a green, brown, and red flag as a ticker to indicate expenses against the budget threshold along with a percentage of each expense category.



## Visual Glance

We finally present to you our Advance prototype app in the short clip below of the additional features we have incorporated into the basic prototype.



0:00 / 1:59



---

[Advanced Prototype](#)

Hope you had a nice time reading through our blog. We like to continue to keep our readers interested, so do visit us back for our upcoming blog on Beta Prototype.



[Home Page](#)