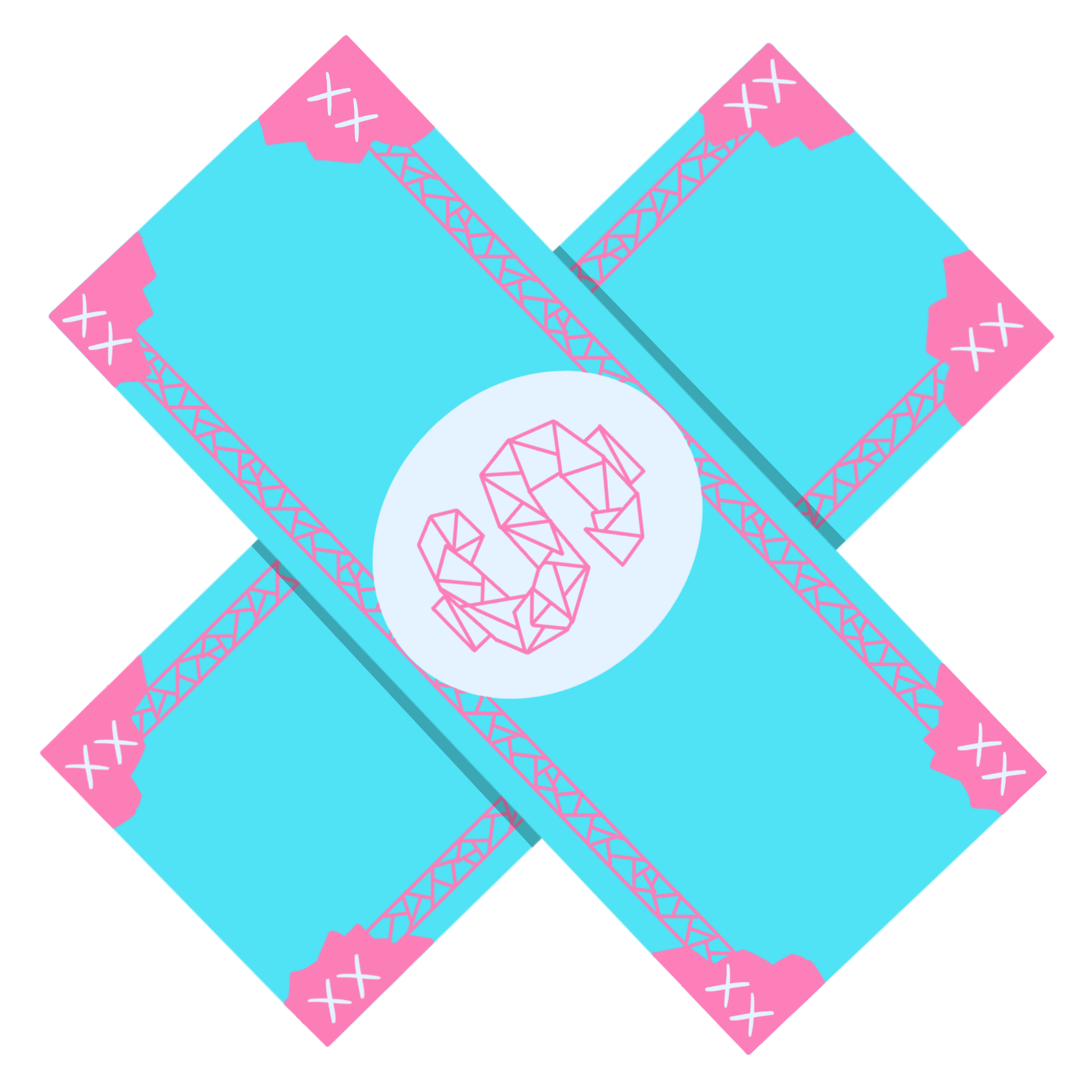
P.A.L.S.S.

Xpendit

**Software Architecture Document**



**Table of Contents**

1. Introduction 3

1.1 Purpose 3

1.2 Scope 3

1. Architectural Requirements 3

2.1 Goals and Constraints 3

2.3 Architectural Views 3

1. Architectural View Decomposition 4

3.1 Use-Case View 4

3.2 Design View 5

3.3 Process View 6

1. Size and Performance 6
2. Quality 8

# Introduction

## 1.1 Purpose

The purpose of this document is to provide a complete overview of the architecture for the application *Xpendit*. The sections of this document contain the various specific information that pertains to the architecture of the application. The main goal of the application is to provide a seamless experience to the user by using the architectural designs laid out within this document.

## 1.2 Scope

This document provides a brief overview of the architecture of the *Xpendit* application designed for Android and iOS. Xpendit is designed by P.A.L.S.S. with the intention to provide a tool to keep track of shared expenses.

# Architectural Requirements

## 2.1 Goals and Constraints

The Xpendit architecture has been developed and designed based some on the following use cases:

1. Be able to create groups with multiple members
2. To keep track shared expenses between
3. Manage a form of payment system
4. Allow the creation of user accounts
5. To allow users to see outstanding debt
6. To allow users to pay and accept any outstanding debt

The philosophy of the system is to be simple and intuitive for the user on the front end, while having a backend that can handle all the functional requirements.

Ultimately, Xpendit requires users to willingly share some public information; regardless, there is also an expectation of privacy. The architecture chosen must adapt and conform to these requirements – for example, while a user should expect to share some financial information like how much money is owed to whom, they should not expect other information like bank account or card numbers to be visible to anyone else. With this in mind, we chose to implement Xpendit with a layered security system, and around a layered architecture which separates the data itself from presentation and data manipulation logic.

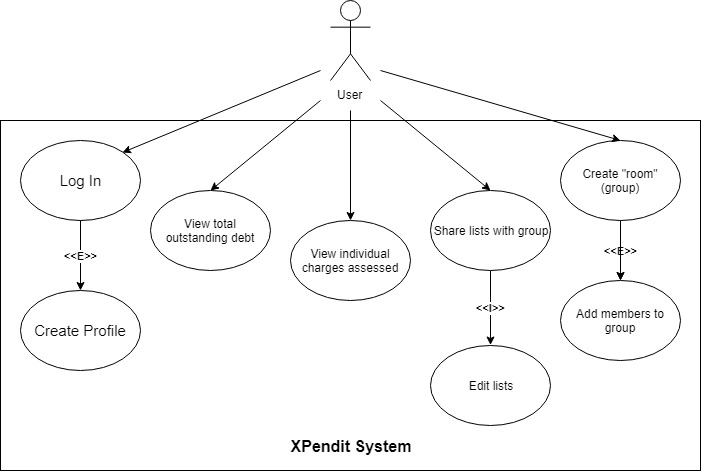
## 2.3 Architectural Views

The architecture will make use of a system of the following views: Use-Case View, Design View, and Process View. Each of the architectural views within this document has been described in detailed and each view can be found in section **3. Architectural View Decomposition**.

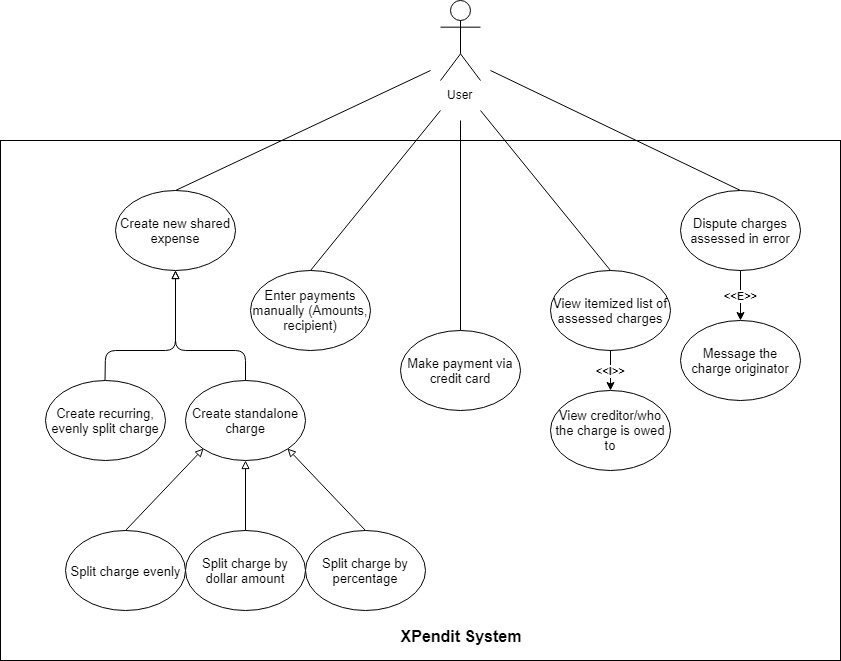
# Architectural View Decomposition

## 3.1 Use-Case View

The following is the Use-Case diagram for the user interface:



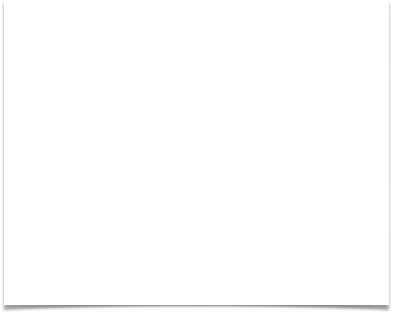
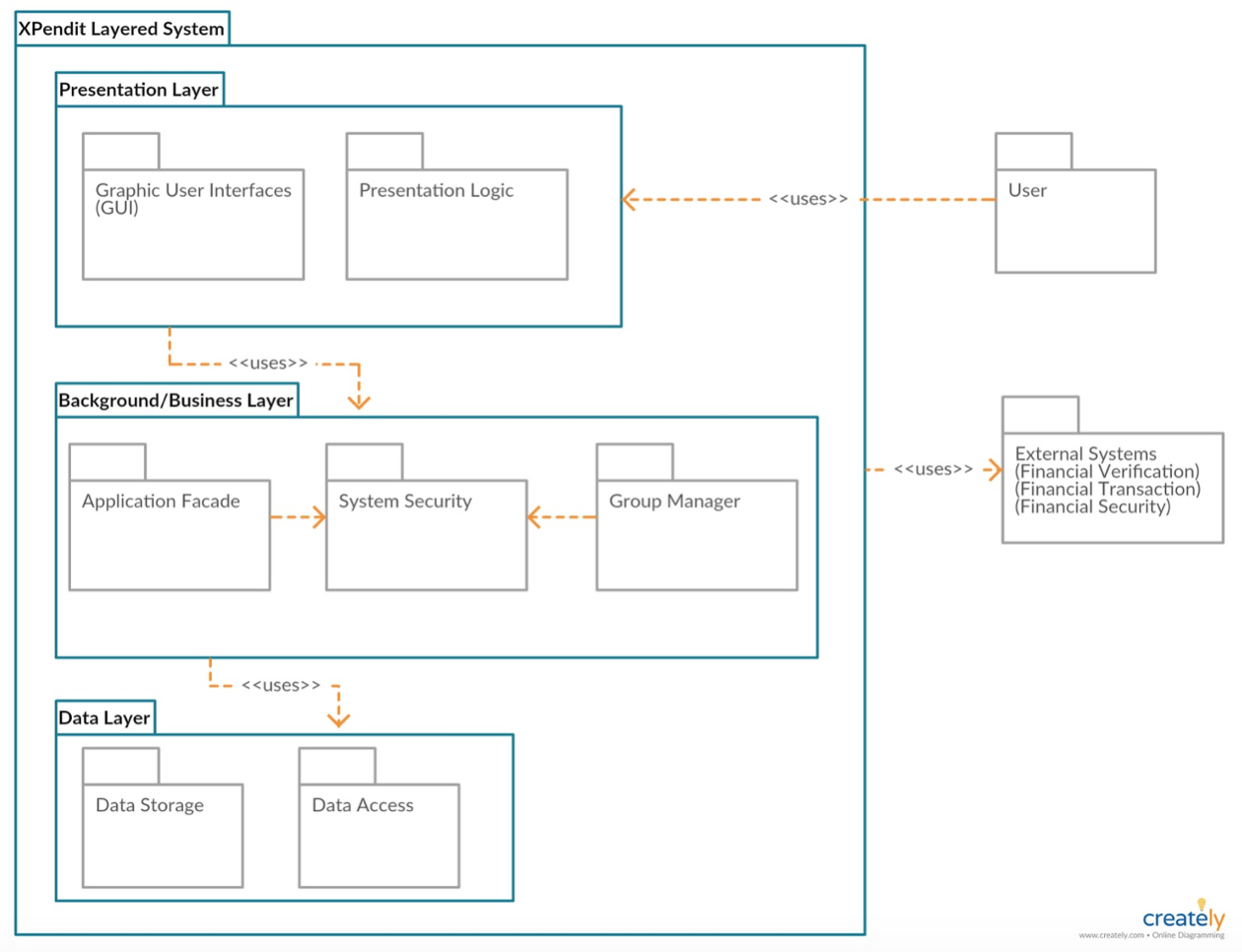
The following diagram is for the Use-Case of the backend for the application:



The interface diagram show the use cases of that the user will go through when interacting with the interface of the application. The diagram should be referenced when designing and implementing the user interface for the application.

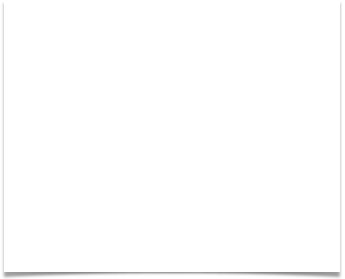
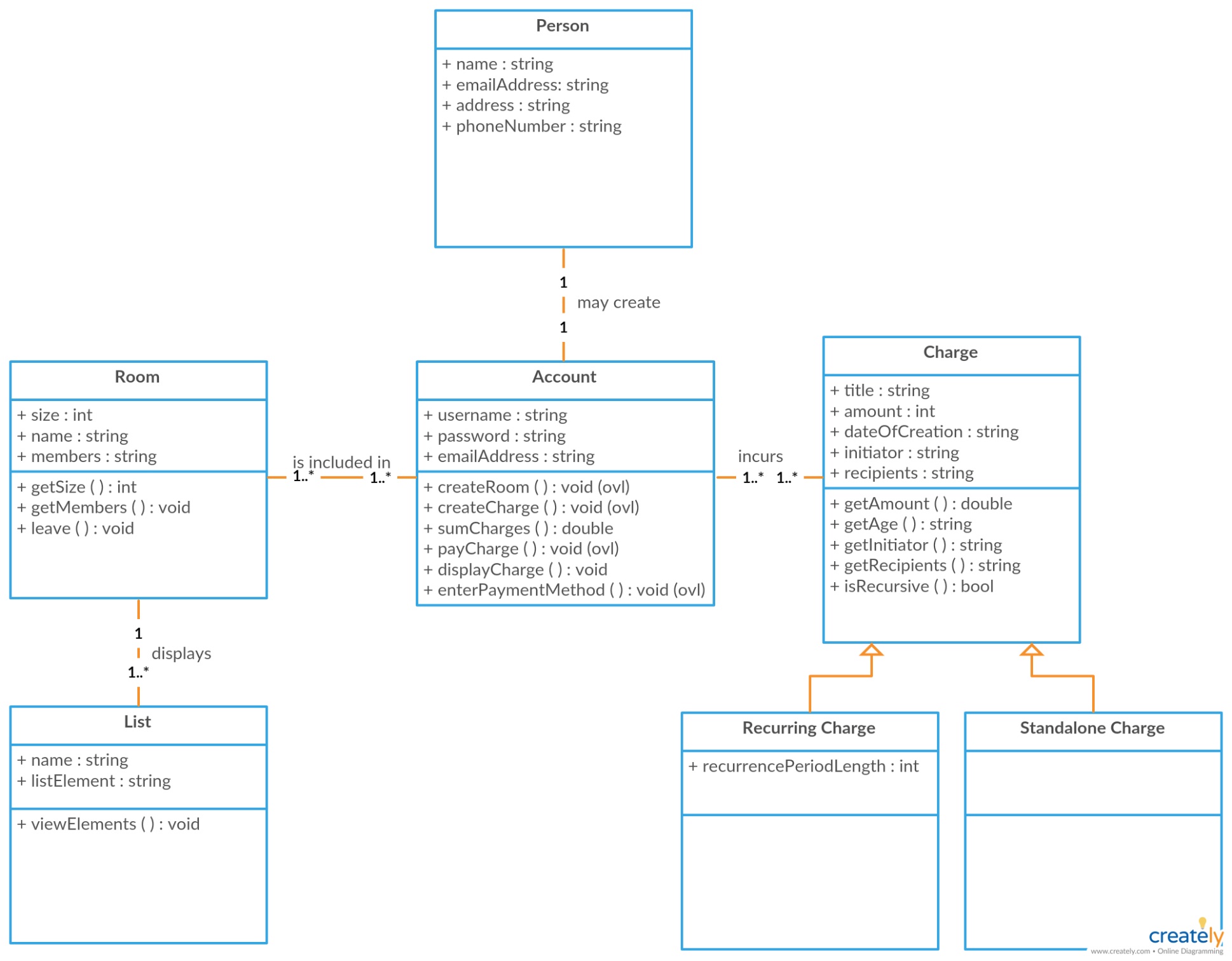
The backend diagram shows the use case for how the system should respond to various use cases. Within the diagram is the structure of how the application and system should be designed to handle the different possible uses that will occur.

## 3.2 Design View



The design view lays out the systems involved within the application. There are three layers within the *Xpendit* system. The first layer is the the presentation layer. This is the layer that the user interacts with and consists mainly of the graphic user interface. The second layer is the background processes. Within the background processes is most of the backend that runs the application. Included within this is the security system that handles transactions with external systems. As well as the management of user balance and groups. The final layer is the data storage. Everything from user accounts to group information will be stored within the database and gets accessed by the background layer.

## 3.3 Process View



The process view shows the classes involved within the application. The first class is the Person class. This is the class that represents the user, from there, that user may create one Account. Once an account is created that account may join/create a Room instance or incur a Charge instance. Each class has a set of attributes as represented in the diagram.

# Size and Performance

The architecture of *Xpendit* must be able to perform all the following task:

1. The system must allow a user to charge debt to one or more users.

2. The system must be able to alert a user if a charge is file against them.

3. The system must allow the user to accept or dispute a debt charged to them.

3. The system must alert the user if a charge they have made is accepted or disputed by the other party.

4. The system must be able to keep track of a user’s debt.

5. The system must be able to add to or subtract from a user’s debt.

6. The system must be able to display a user’s debts, along with who the debt is owed to and the time since the debt was applied.

7. The system must be able to sort the debts by amount owed, person the debt is owed to, the time since the debt was applied, and by any user defined priorities.

8. The system must allow the users to define their own priorities to be used when sorting debts.

9. The system must allow users to mark debts as paid with the acceptance of the other person.

10. The system must allow the user to attempt create an account with a username or password.

11. The system must be able to check if a proposed username is unique or not.

12. The system must only accept the account if the proposed username is unique

13. The system must allow the user to log into an account if they are not currently logged in to an account.

14. The system must allow the user to log out of an account if currently logged in.

15. The system must allow users to create rooms containing themselves and other users

16. The system must allow users in the room to add other users not currently in the room.

17. The system must allow users without outstanding debts to remove themselves from a room

18. The system must allow a room to be deleted if no outstanding debts are owed between any of the members in the room.

19. The system must allow users in a group to view the bulletin board for any group they are in.

20. The system must allow the user to create posts in the bulletin board

21. The system must allow the user to comment on posts that have been created in the bulletin board.

22. The system must allow users in a group to see the shared shopping list in a group.

23. The system must allow users to mark that they have bought an item in the shopping list.

24. The system must apply the charges to the person corresponding to the item in the shopping list that is bought.

25. The system must allow the user to delete items from the shopping list that they added themselves.

26. The system must automatically remove ought items form the shared shopping list.

The client-side of the architecture will be run on either an Android or iOS device.

# Quality

The software architecture has been designed to be able to support the following specifications:

1. The system should be able to run anywhere where there is a valid internet connection that can access the server database.

2. The application should be able to achieve a start time less than 20 seconds in 90% of cases.

3. The application will require the user to make a strong password of multiple characters and longer than 10 characters

4. The software will be perfectly correct in keeping track of debts owed.

If these requirements are not met, then the system will still be capable of functioning.