Architectural and Design Report

Group: Shahd Mustafa, Mike Nasser, Nathaniel Leonardo, Hassan Radwan, Hussen Al-Jubury

Table of Contents:

1.0 Introduction

1.1 Purpose

- This document will provide an overview of the architecture of our planned splitsmart application using a variety of different use cases and diagrams. We hope to convey our main goals, objectives and challenges for this project.

1.2 Scope

- This Architecture Document applies to the splitsmart expense sharing application to be developed by our term project team.

2.0 Architectural representation

This document presents the architectural design as a series of views, including use case view, logical view, process view and deployment view. These are presented as diagrams with accompanying explanations that implement the Unified Modeling Language (UML).

3.0 Architectural goals and constraints

3.1 Architectural Goals

1. Scalability: The software should be designed to handle a large number of users, groups, and expenses efficiently. It should be capable of scaling both vertically (increasing server capacity) and horizontally (adding more servers) as the user base grows
2. Reliability: The software should be reliable and available to users, minimizing downtime and ensuring data integrity. Measures such as backup and recovery mechanisms should be implemented to prevent data loss
3. Security: The software should prioritize the security of user accounts, sensitive information, and transactions. It should incorporate authentication, authorization, and encryption mechanisms to protect user data and prevent unauthorized access.
4. Usability: The user interface should be intuitive, easy to navigate, and provide a pleasant user experience.

3.2 Architectural restraints

1. Platform compatibility: The software should be designed to be accessible through multiple platforms, such as web browsers and mobile apps, to cater to a wider user base and provide flexibility in accessing the system.
2. Performance: The software should be optimized for responsiveness and efficient data processing. Actions like creating expenses, updating balances, and generating reports should be performed in a timely manner to ensure a smooth user experience.

3. Integration: The software may need to integrate with external services or APIs for features such as receipt scanning, payment processing, or notifications.The architectural design should accommodate these integrations effectively.
4. Compliance: Depending on the region or industry requirements, the software may need to adhere to specific regulations regarding data privacy, financial transactions, or other legal considerations. Compliance with applicable standards should be incorporated into the architecture

4.0 User case view

Defining use cases is essential for designing a system, as they illustrate how the system will be utilized and which functionalities it should provide. They serve to connect the design of the system with its intended purpose by outlining particular interactions and behaviors from the standpoint of users or stakeholders. By identifying and documenting multiple use cases, architects can document the requirements, limitations, and interdependencies of the system, which then form the foundation of the overall architecture.

The use cases in the system are listed below (**bold** use cases are important to the architecture).

- **User login**
- User registration
- **Payment information**
- Manage groups
- **New expense**
- Split expense
- Assign expense
- **Customer report**
- Expense notification
- **Payment**
- **Homepage**

The following diagrams show the use cases within the system.

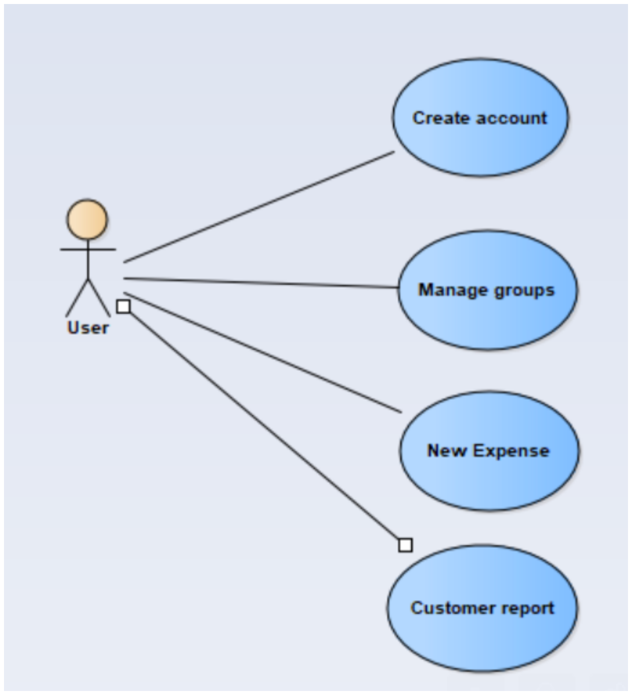


Figure 1: user account use case

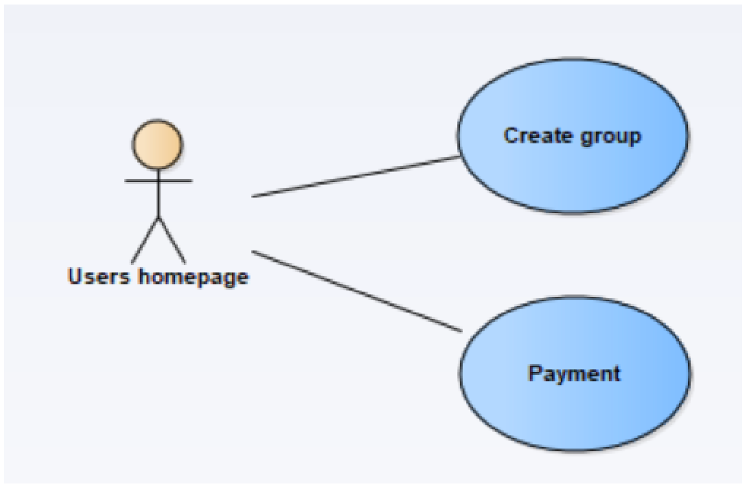


Figure 2: creating expense use case

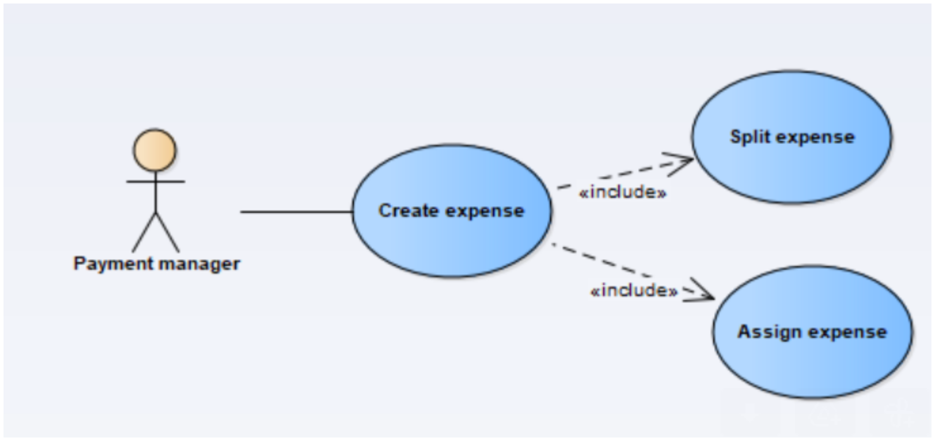
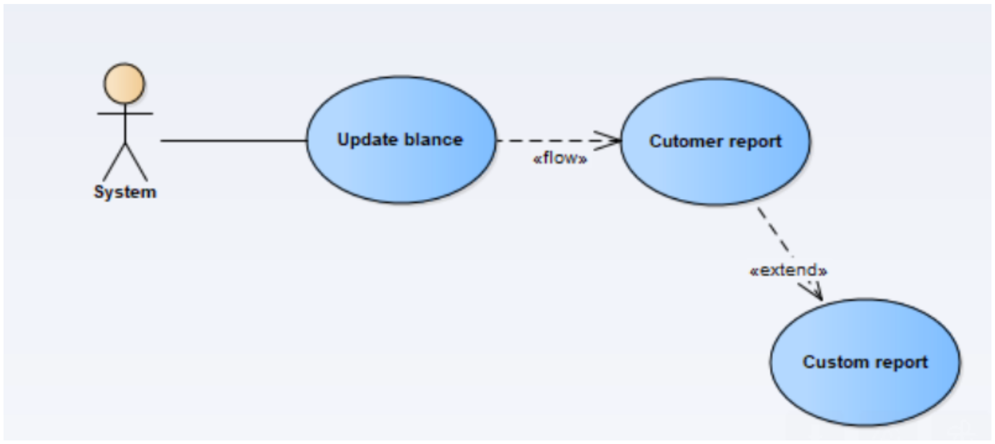


Figure 3: balance update use case



4.1 significant use case descriptions

- User login:
  This use case occurs when the customer accesses the split smart site and wishes to access their account, by entering their username and password.thus they are directed to the home page.
- User registration:
  This use case occurs when a new customer creates/register an account by entering the following information to the system: name, address, phone number, email address, username and password.Thus an account will be created.
- Payment information:
  This use case occurs when the customer is entering their profile information and specifying their credit card number and pin. This could also be updated when the user is making new payments.
- Manage groups:

This use case occurs when the user wants to create a group by entering the name and email of members, and naming the group. An invite will be sent to valid members to join the group.

- New expense:
  This use case occurs when the customer wishes to create a new expense by specifying the amount, date, description, split and optional image.
- Split expense:
  This use case occurs when a new expense is made by the customer to enter the split percentage.
- Assign expense:
  This use case occurs when a new expense is made by the customer to assign the expense to members of the group.
- Customer report:
  This use case occurs when the customer wishes to view the payment report or a custom report by entering in a date.
- Expense notification:
  This use case occurs when a new expense is made, a notification is sent to members that have been assigned to an expense.
- Payment:
  This use case occurs when the customer wishes to make a payment by specifying which and using their pre-registered account information to pay. The balance is updated for the customer and the group's balance too.
- Homepage:
  This use case occurs when the user is logged in to the system to view the balance information, joined groups, reports, friends and their profile information.

5.0 Logical view

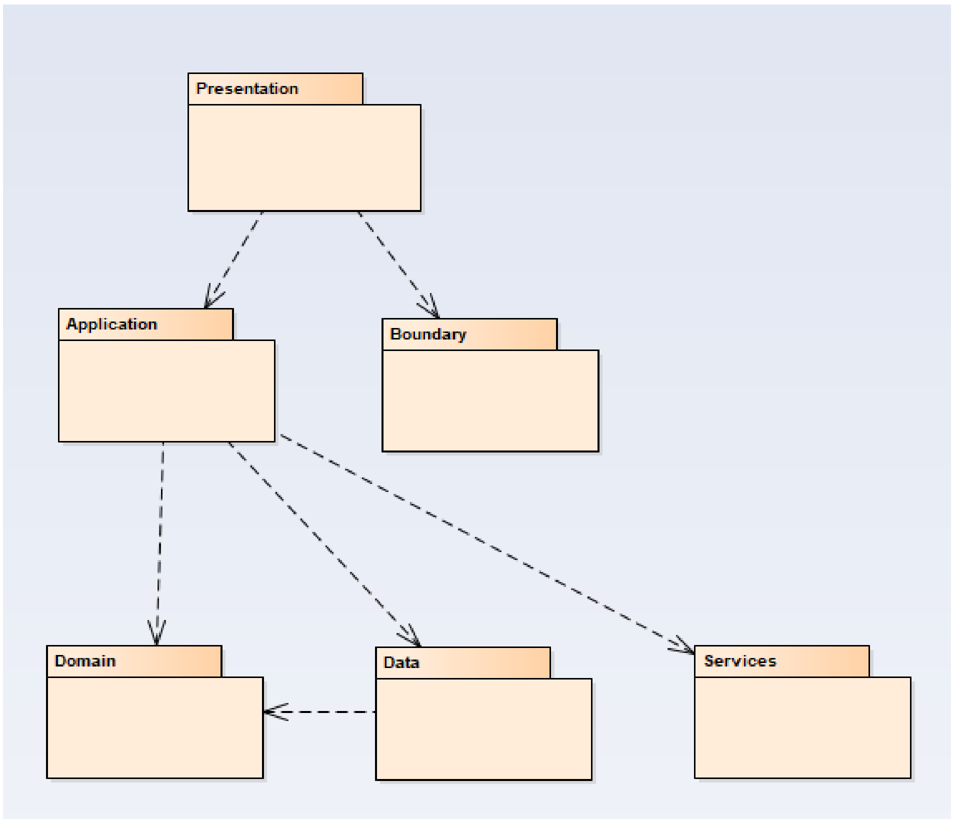
5.1 Overview

The logical view of software architecture is crucial in software engineering because it facilitates a comprehensive comprehension of how various system components interact and are organized to achieve the desired functionality. It promotes scalability and modularity while enabling effective communication and better system analysis and design.

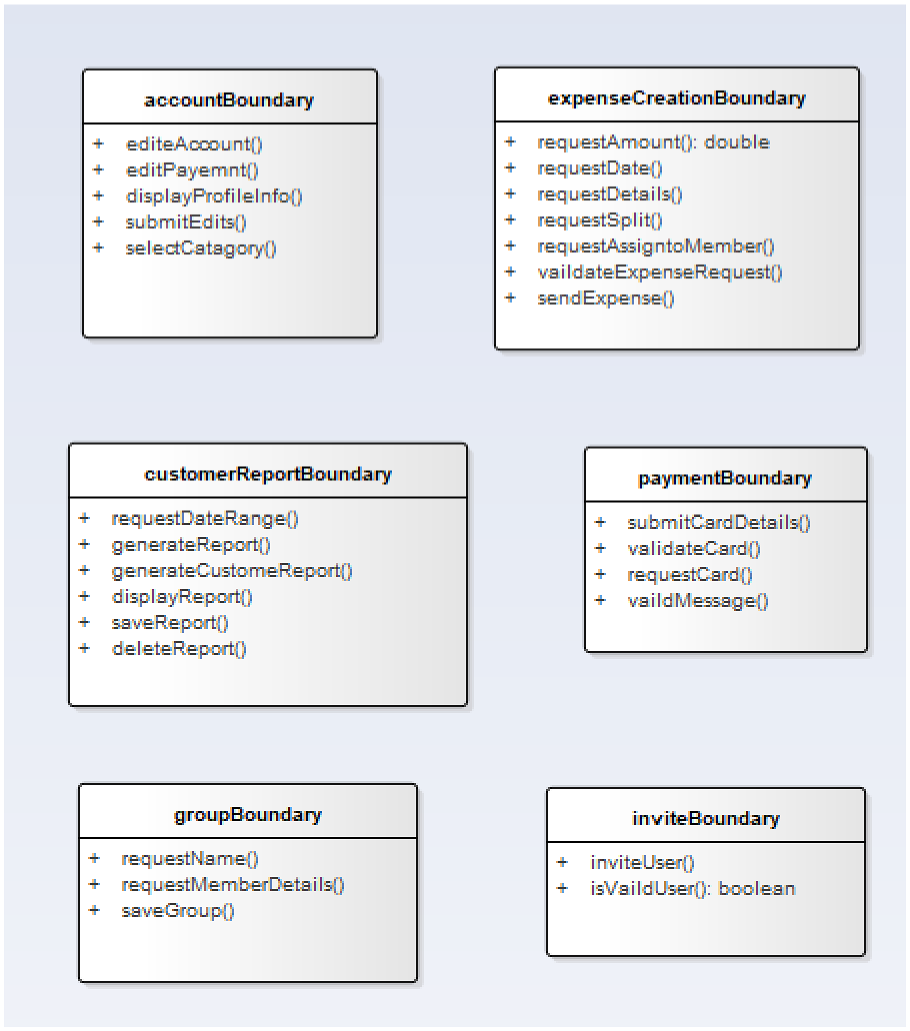
- **Presentation:**
  - This package contains classes that handle the user interface and forms including the user login, expense and group creation, generating reports and payments.
- **Boundary:**

- This package includes classes that support the interactions between the user and system, group actions, user profiles, expense creation, managing payments, reports and payment processing.

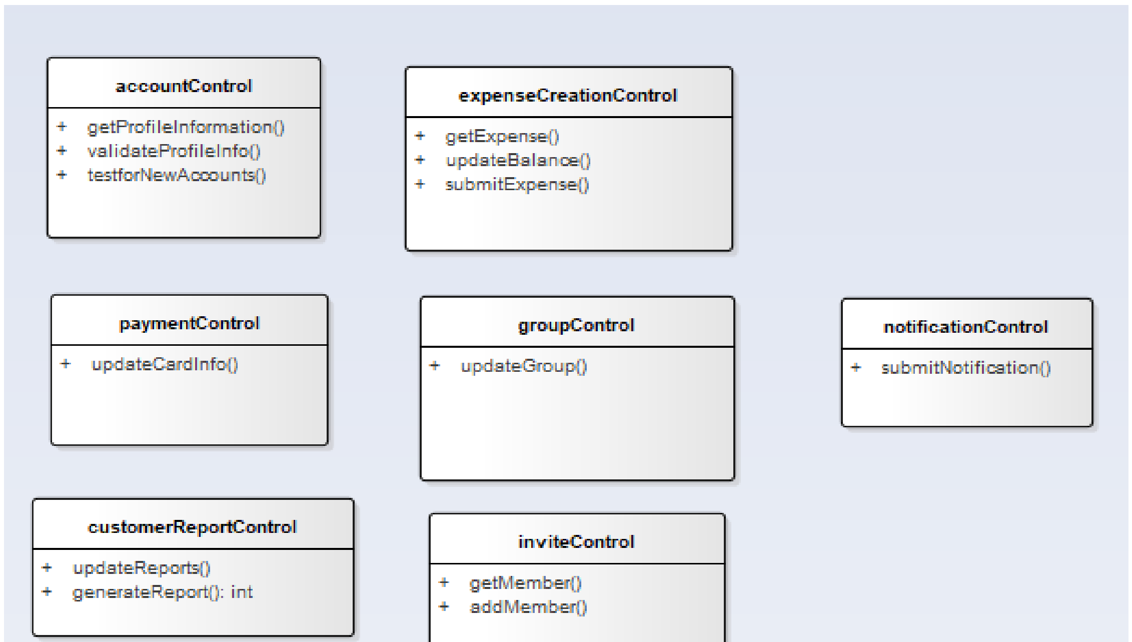
- **Application:**
    - This package handles classes that enable users to interact with the system, manage user profiles, facilitate group-related actions, handle expenses, generate payment reports, and process payments.
- **Domain**
    - This package contains classes that encapsulate the core entities and logic of the system. It includes classes for users, groups, expenses, reports, and payment processing.
- **Data:**
    - This package includes classes responsible for persisting data within the system, such as storing user profiles, group information, expense details, and payment records.
- **Services:**
    - This package handles classes that provide system-level functionality for maintenance purposes. It includes classes for manual maintenance tasks, such as backup and restore, system configuration, and user administration.



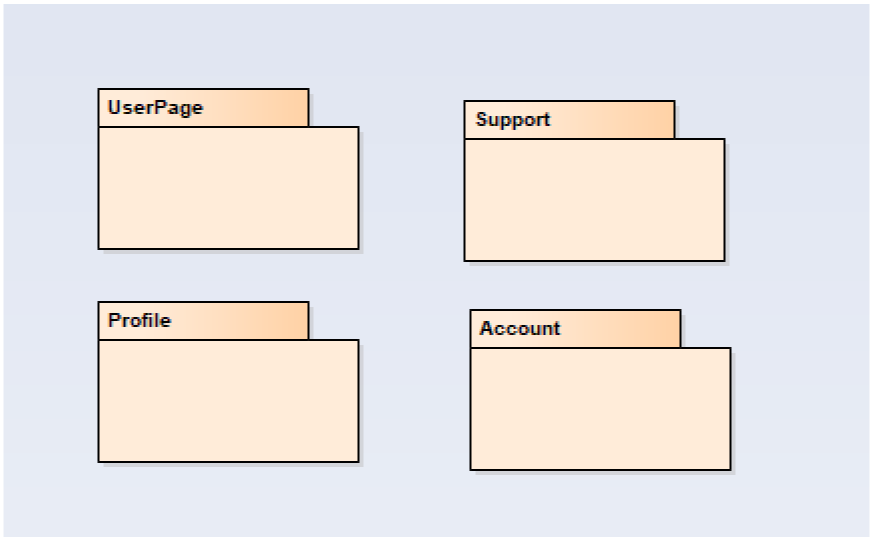
Logical View



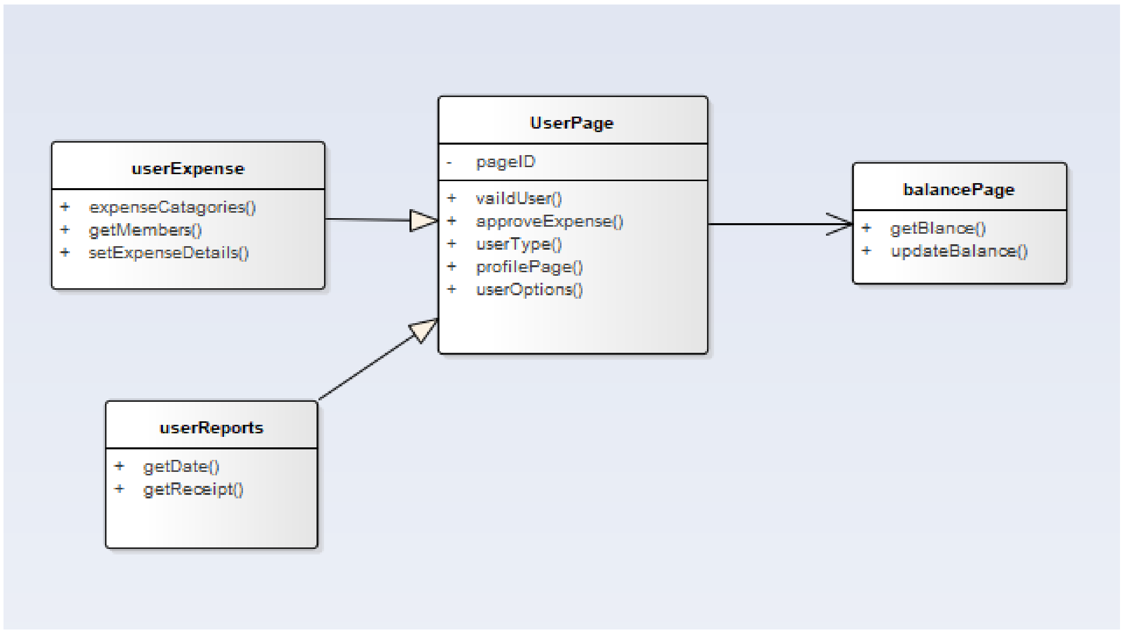
Presentation Package



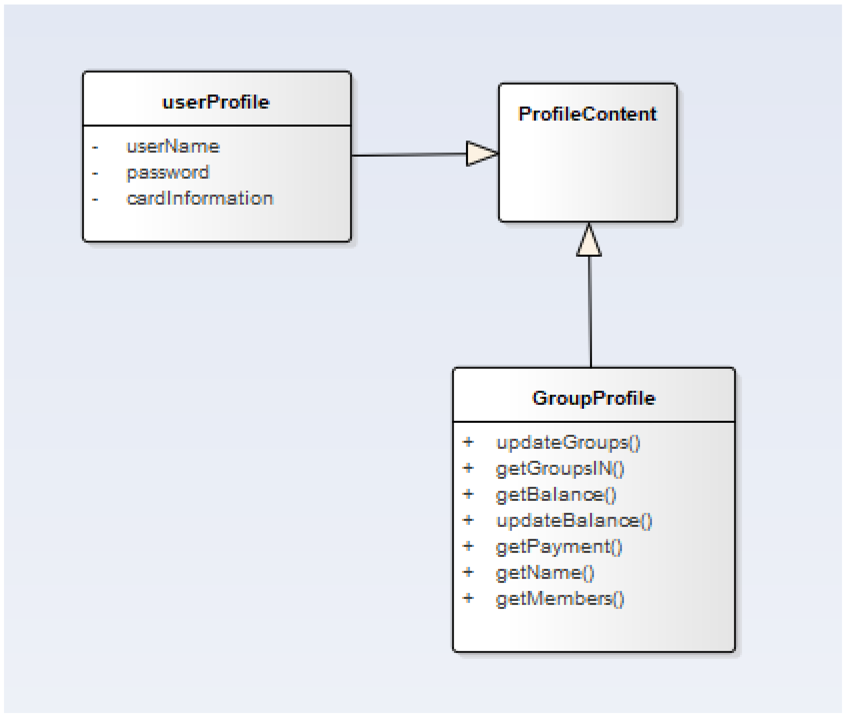
Application Package



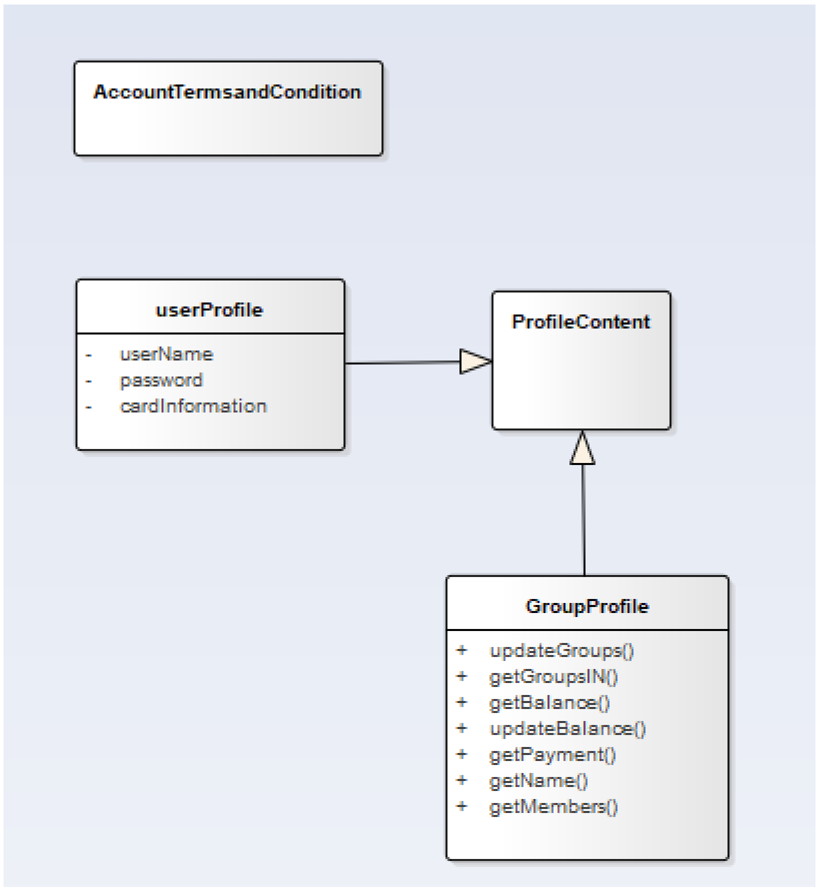
Domain Package



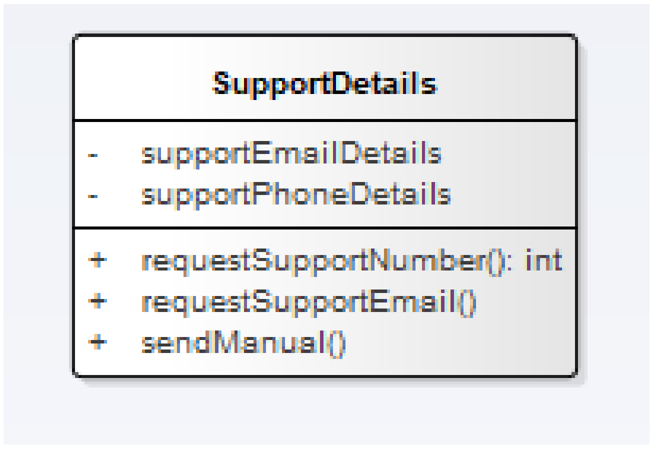
User Page Package



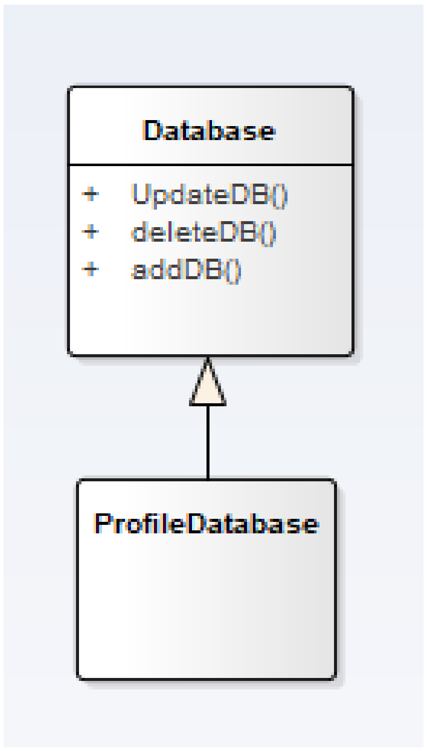
Profile Package



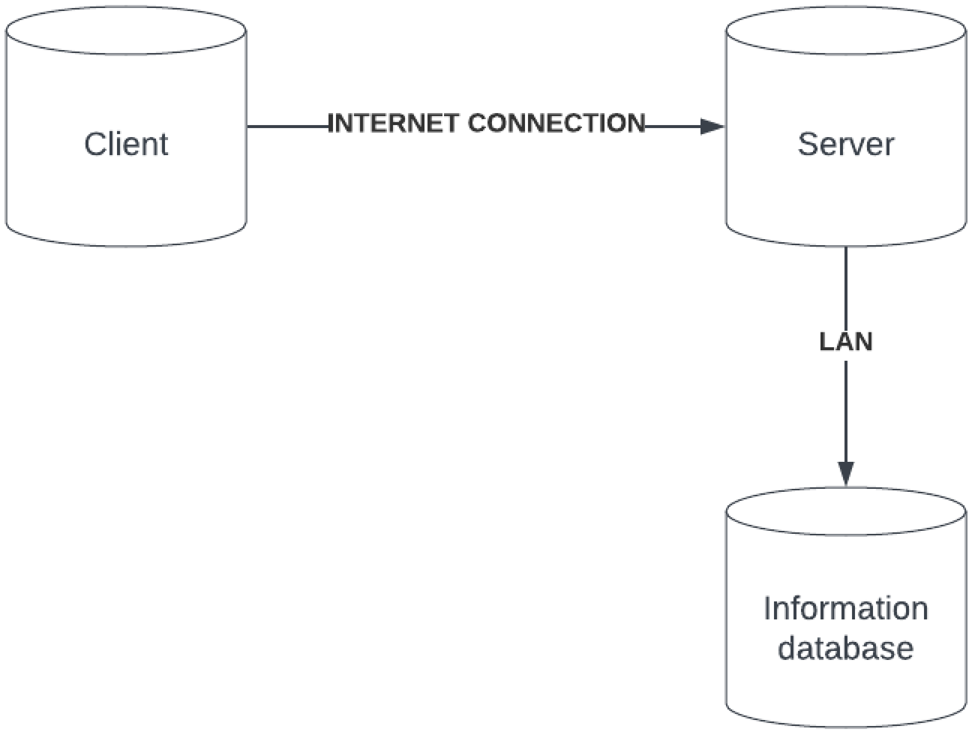
Account Package



Support Package



6.0 Deployment view



The client represents any client device with an internet connection, whether it be PC or Mobile application. This user then connects to the server and obtains information from the database for the applications usage

7.0 Size and performance

Currently the software is designed to support roughly any number of users at the local level simply by having the application, however in regards to the online payment and login features we expect to be able to sustain roughly 10,000 users simultaneously.

8.0 Quality

The Splitsmart application aims to provide a high-quality user experience through a well designed architecture. It prioritizes reliability by minimizing downtime and incorporating fault tolerant mechanisms. Security measures ensure the protection of user data and transactions. Scalability and performance are addressed through the ability to handle number of users and efficient system design. Maintainability is supported by modular components and clear documentation. Usability is enhanced through an intuitive interface, while extensibility enables future enhancements. Testability ensures the application's correctness and reliability. By considering these quality attributes, the Splitsmart application strives to deliver a reliable, secure, scalable, and user friendly expense sharing solution.