Department of Computer Science-Software Engineering

Software Design

Lab Midterm

COMSATS University Islamabad

Dhamtor campus

Submitted to:
Sir Mukhtiar Zamin

Submitted by:
Khizar Ali
Sp23-bse-047

CASE STUDY ARCHITECTURE

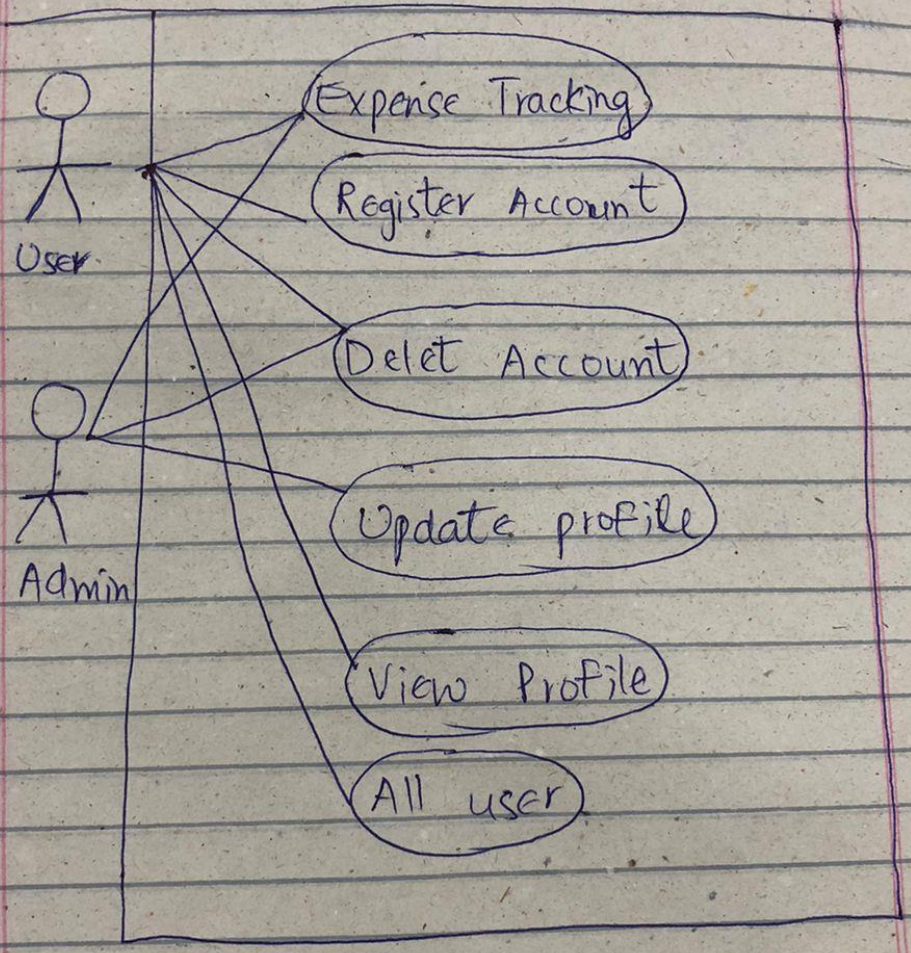
SMART TRAVELLING MANAGEMENT SYSTEM

USECASE DIAGRAM:

Khizar Ali SP23-BSE-047

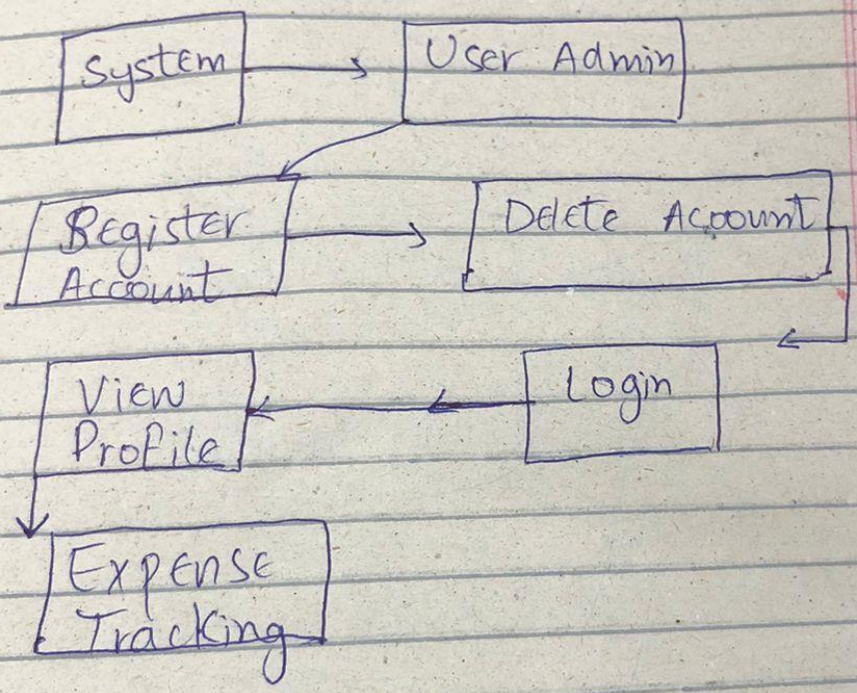
USECASE DIAGRAM

SMART TRAVELING MANAGEMENT SYSTEM



KhizarAli SP23-BSE-047

COMMONICATION DIAGRAM



Description of the Observer Pattern Implementation for "View Profile" Use Case

This Java code demonstrates the **Observer Pattern** in the context of a "View Profile" use case. The **Observer Pattern** is a behavioral design pattern where an object (the *subject*) maintains a list of its dependents (the *observers*) and notifies them automatically of any state changes, usually by calling one of their methods. In this case, we simulate how an updated user profile (the subject) triggers changes in multiple observers that need to reflect this updated profile information.

Key Concepts in the Code:

1. **UserProfile (Subject)**:
    - The `UserProfile` class represents the "subject" in the Observer pattern. This class holds the profile data, specifically the `username` and `email` fields.
    - It provides methods for registering and removing observers (`addObserver()` and `removeObserver()`), as well as notifying them when the profile data changes (`notifyObservers()`).
    - When the profile details are updated (such as changing the `username` or `email`), the `UserProfile` class calls `notifyObservers()` to inform all registered observers about the change. This ensures that the observers are always in sync with the latest profile data.
2. **ProfileObserver (Observer Interface)**:
    - The `ProfileObserver` interface defines the contract that any concrete observer must adhere to. It contains a single method `update(UserProfile userProfile)` that will be called when the subject (UserProfile) changes its state.
    - This allows different types of observers to react in their own way to profile changes, ensuring flexibility.
3. **Concrete Observers (ProfileView and ProfileDashboard)**:
    - **ProfileView** and **ProfileDashboard** are concrete classes that implement the `ProfileObserver` interface.
    - These classes represent the "observers" that listen for changes in the `UserProfile` and respond accordingly by updating their views or performing other actions.
    - Each observer has its own `update()` method, which outputs the latest profile information (`username` and `email`) to simulate how a profile view might update when the data changes.
4. **Main Class (Test the Observer Pattern)**:
    - The `Main` class sets up the system to demonstrate the Observer pattern.
    - It creates a `UserProfile` object (subject), then creates and registers two observers (`ProfileView` and `ProfileDashboard`).
    - It updates the user profile (e.g., changing the `username` and `email`) and shows how both observers are automatically notified of these changes and update their output accordingly.