Technical University of Moldova
Inginerical department S.A.

Report

№: 5

AMOO

Subject: Class diagrams & SWOT analysis

Author: Terman Emil FAF161
Prof: M. Gavrilita

Chisinau 2018

1 Tasks

– model your application using Class Diagrams;
– perform the SWOT analysis of your project;

2 Theory

2.1 Class diagrams

Also known as structural diagrams, represent the static view of an application. Class diagrams describe the attributes and operations of a class and also the constraints imposed on the system. They are widely used in modeling object-oriented systems, because they are the only UML diagrams, which can be mapped directly with object-oriented languages. Because of that, they are widely used at the time of construction. These diagrams show a collection of classes, interfaces, associations, collaborations and constraints.

Class diagram's main purpose:
– describe responsibilities of a system;
– forward and reverse engineering;
– base for component and deployment diagrams;
– analysis and design of the static view of an application;

Possible relations:
– **Dependence**: relation where one entity is dependent and another independent;
– **Association**: represents a family of links. A binary association (with two ends) is normally represented as a line;
– **Generalization**: indicates that one of the two related classes (the subclass) is considered to be a specialized form of the other (the super type) and the superclass is considered a Generalization of the subclass;
– **Realization**: is a relationship between two model elements, in which one model element (the client) realizes (implements or executes) the behavior that the other model element (the supplier) specifies;

2.2 SWOT analysis

It is a strategic planning technique used to help a person or an organization to identify the Strengths, Weaknesses, Opportunities and Threats related to business competition or project planning. Strengths and Weakness are frequently internally-related, while Opportunities and Threats commonly focus on environmental placement.
– ***S****trenghs*: characteristics of the business or project that give it an advantage over others;
– ***W****eaknesses*: characteristics of the business which can place the business or project at a disadvantage relative to others;
– ***O****pportunities*: elements in the environment that the business or project could exploit to its advantage;
– ***T****hreats*: elements in the environment that could cause trouble for the business or project;

2.3 Design principles

Cohesion and **Coupling** deal with the quality of an OO design. Generally, a good OO design should be loosely coupled and highly cohesive. The aim of the design should be to make the application: easier to develop, maintain, extend and for it to be less fragile to changes.

Cohesion is used to indicate the degree to which a class has a single, well-focused purpose. Coupling is all about how classes interact with each other, on the other hand cohesion focuses on how single class is designed. Higher the cohesiveness of the class, better is the OO design.

CouplingCoupling is the degree to which one class knows about another class. Let us consider two classes class A and class B. If class A knows class B through its interface only i.e it interacts with class B through its API then class A and class B are said to be loosely coupled. If on the other hand class A apart from interacting class B by means of its interface also interacts through the non-interface stuff of class B then they are said to be tightly coupled. Suppose the developer changes the class B's non-interface part i.e non API stuff then in case of loose coupling class A does not breakdown but tight coupling causes the class A to break.

Concerns are the different aspects of software functionality. For instance, the "business logic" of software is a concern, and the interface through which this logic being used is another. The separation of concerns is keeping the code for each of these concerns separate. Changing the interface should not require changing the business logic code, and vice versa. Model-View-Controller (MVC) design pattern is an excellent example of separating these concerns for better software maintainability.

Information hiding is the process of hiding details of an object or function. Information hiding is a powerful programming technique because it reduces complexity. One of the chief mechanisms for hiding information is encapsulation - combining elements to create a larger entity. The programmer can then focus on the new object without worrying about the hidden details. In a sense, the entire hierarchy of programming languages, from machine languages to high-level languages, can be seen as a form of information hiding. Information hiding is also used to prevent programmers from intentionally or unintentionally changing parts of a program.

Conceptual integrity is the principle that anywhere you look in your system, you can tell that the design is part of the same overall design. This includes low-level issues such as formatting and identifier naming, but also issues such as how modules and classes are designed, etc. This is vitally important because inevitably unanticipated issues come up that must be resolved quickly.

3 Class Diagram

In *Figure 1* it's represented the general structure of Facebook that is related to the user. It contains:
– Account;
– Profile;
– Converstaion;
– Message;
– Comment;
– RCSContent (React Comment Share content)
– Photo;
– Share;
– Reaction;
– Account State interface;
– Active account state;
– Banned account state;
– IssuedContent interface;

The most important class, is the **Profile**. It has an *Agregation* from **Account**. (Why not *Composition*? Because the account may get deleted, but the profile may still remain. The profile is also used to identify comments, reactions, shares, converstations and other things). All the user's friends are stored inside the Friends list. FriendRequests list is a container for pending friend requests. The user may *ProcessFriendRequest()* and set it as accepted or rejected. The profile may be also deleted. In this case, all its public *Issued content* will become anonymus.

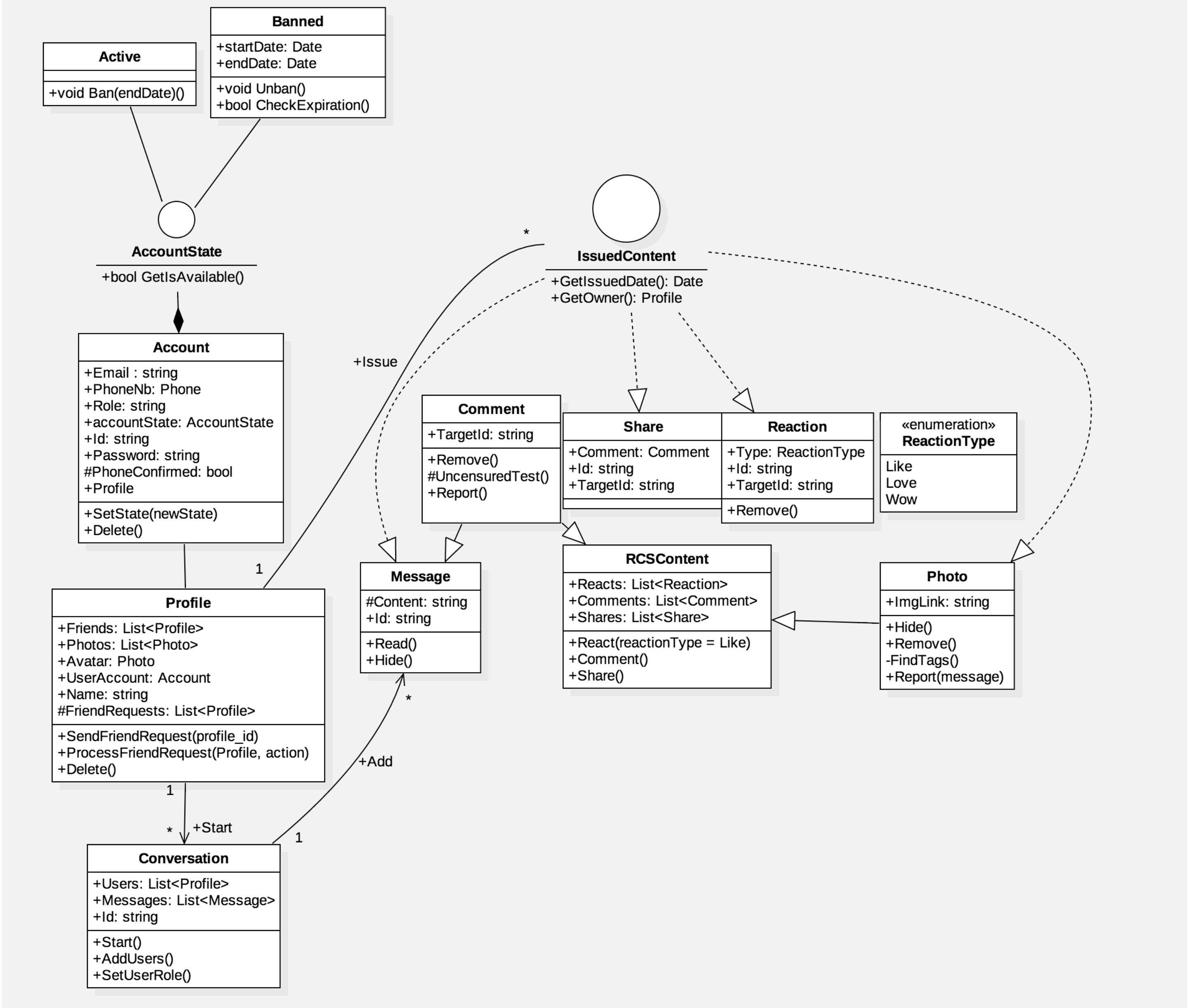
The **Account** class contains all the user's identity. Some of its fields may be set to public in the profile settings. The boolean which indicates if the phone is confirmed is used when the user tries to LogIn with his phone number. The Role is used to diffirenciate between a simple or a special account, like an Admin. Moving on to *Account state*: this represents an interface which shows if the account is available. For example, when some people try to find a user A, before they can see him, **accountState.isAvailable()** is checked first.

AccountState is an interface which is realized by **Active** or **Banned** classes. When a user is banned by an admin or multiple reports, an expiration time is set. If he is banned undefinetly, the *endDate* is set to its maximum value.

IssuedContent is an interface assigned to anything that has an Owner and a creation time.

RCSContent is a base class for anything that can be Commented, Reacted or Shared.

The **Comment** class inherits from both *Message* and *RCSContent* and has a protected opperation which checks if the content is Uncensured. If there are problems, the comment is hidden until a special user can approve. It also has the option to be reported by anyone who sees the comment, which may lead to banning the owner's account. The *TargetId* is the Id of the content that the comment was issued to. Any content on this platform has a unique ID. As said before, this class inherits from **Message**. This class has a protected



Img 1: **MiniFacebook** class diagram - user related

string which represents the content. If the message contains a link to a video, or an image, or it's an audio/video message, then *Read()* opperation will properly display it.

4 SWOT Analysis

1. **Strength**:
   - saves a lot of time;
   - provides a large varity of configurations;
   - user friendly interface;
   - the project does not require our office to be in a specific region;
2. **Weaknesses**:
   - there are content restrictions;
   - accounts may get banned;
3. **Opportunities**:
   - user feedback will greatly help with the application development;
   - during the recent incident with privacy at Facebook, many users will try to find a more secure platform;
4. **Threats**:
   - big fishes like Twiter and Instagram may be enough to saturate the market;
   - little budget means we can be easily manipulated;

5 Conclusion

In this laboratory work, I got a very good grasp of what are Class Diagrams and how they are useful. During this lab, I learned more about all the Design Principles mentioned before. While implementing the UML diagram, I asked myself over and over if my classes are Cohetic enough and how independent are they. It forces the developer to think one more time about the structure, which is a huge help for later development: in my opinion, it's better to think very careful about the structure first, than refactoring the app later. It also helps the developer to better see what he can generalize and what can be isolated. I am sure that from this laboratoy on, I will start using Class Diagrams for my projects.