Social Networking Platform Using C++

Object-Oriented Programming

Ahmed Elsaid

May 2, 2022

New York University Abu Dhabi

1 Introduction

1997 was the year when the first social media platform ever SixDegrees.com was launched [1]. People could create profiles of themselves, make connections with other people, and exchange messages within networks [1]. Since then, with the launch of Facebook, masses of people have been attracted to the so-called social media age. Before that, it would have been "ridiculous" or "out of this world" to think that the world can be connected on one platform with just a few clicks on keyboards or touches on screens. Innovations have been rolling out consistently since then, and the dawn of shared virtual environments (Metaverse, Decentraland, etc.) seems to be upon us soon.

The most fascinating thing about today's social media platforms is how the companies responsible make it a seamlessly simple experience for people from all walks of life. With this, companies have been able to achieve the goals of inclusivity, diversity, and accessibility on their platforms [2]. However, with this comes advantages and disadvantages for these platforms, which entails a huge responsibility for the development companies. Furthermore, topics like the security of data and information, and censorship and regulation of content must be considered seriously by the developers.

In 2021, it was recorded that there were 4.8 billion in users of social media platforms [2]. This is a growth of 13.13% year-on-year from 2020, which shows how social media is still growing rapidly despite it being around for more than 20 years [2]. Back in 2015, there were only 2.07 billion users of social media, so that is double the number in only 6 years [2]. The ongoing growth of social media proves that there are unimaginable opportunities for development and innovation in the social networking space.

In this paper, I will be discussing the approach and the solution that I took to build a simplistic social media platform using the C++ programming language.

2 Approach

I approached the problem systematically and tried to do it in an organized way. I divided the process of building my platform into three main stages: ideation, preparation, and execution.

Ideation

First, at the beginning of the ideation stage, I was mainly thinking informally without writing anything on paper. I did my due diligence and made some research by studying platforms that I would like to be inspired by. I explored platforms like Facebook, Instagram, Twitter, and LinkedIn. While looking at these platforms, I was noting down the functionalities that they have on their platform. This gave me an idea about the structure that I want to have, the flow of my platform, and the functionalities that my platform can perform. Then, I moved on to the preparation stage.

Preparation

In the preparation stage, I started by eliminating the functions that I could not include because of time constraints and current knowledge. Then, I wrote them down in my notes. I also prepared the classes that I want to have and their child classes. Furthermore, I listed down the Object-Oriented Programming concepts that I wanted to use in my platform and noted down the practices that I wanted to stick to when writing my code. Then, I drew a diagram of how the program will be structured. Afterward, I moved on to the last and most important step in my project.

Execution

Finally, came the execution part where I had to start writing code. I used the C++ programming language to write my code. I wrote my code carefully by following best practices that I wanted to maintain and by using object-oriented concepts. I tried to keep my code as simple as possible and not include anything that I do not understand. After finishing each part, I made sure to comment out the code rigorously to make it easy for someone reading my code and to make it easy for me to pick up my work after taking a break.

3 Solution

Before starting to write my code, I wanted to create 2 pseudo-code-like structures to follow while coding out my program. I started by creating the class diagram (which can be found

in figure 1) that is composed of one base class "Account" and 2 child classes "User" and "Page". I thought about this class structure after studying some social media platforms and figuring that most of them identify accounts as either pages or users.

Then, I came up with a flowchart that covers what the project performs on a very simplified level. The flowchart diagram can be seen in figure 5.

Object-Oriented Paradigms

I focused on including Object-Oriented programming concepts in my code because this project's performance can be enhanced using an object-oriented approach. First, I used Classes as can be seen in classes Account, User, and Page in figures 2,4, and 3. I then used default constructors, parameterized constructors inside these classes as seen in figures 6 and 7. Then, I used setters and getters to help me with writing other complex functions in my code as seen in figures 8 and 9. When structuring my code, I decided to use inheritance of classes to help my two child classes with having common variables and to share the functions between my derived classes and the base class as seen in figures 4 and 3. Moreover, I utilized virtual functions in my base class while overriding functions in my inherited classes to tell the compiler in advance that the functions in the base class may be overridden by functions in the derived classes (figure 10. I am specifically using virtual functions where applicable because of the use of maps, which requires consistency when calling the functions. This helps with avoiding calling functions from different derived classes (confusion between the User class and Page class).

Optimization Techniques

In addition, I used some optimization techniques to improve the speed of my program and better handle my program's memory management. First, I used virtual destructors inside my parent and child classes as can be seen in figures 11, 12, and 13. The use of virtual destructors in our code helps to avoid memory leaks when data is not automatically released by the compiler. Making the base class destructor virtual guarantees that the object of a derived class is destructed properly [4]. Then, I used inline functions inside the classes to help with runtime performance optimization when I wanted to do simple operations like addition [6] (figures 14, 15, and 16). I also ensured that my code is modular as possible to

shorten my main function. Modular codes help with avoiding rewriting existing code, which is helpful when writing big programs; thus, modular programming does benefit memory handling on big projects. Furthermore, modular code gives the programmer, or whoever the project is handed over to, ease of debugging and modifying the code. Using modular programming standards, the code is kept short, simple, and easy to understand. In addition, I postponed my variable declaration as long as possible and declare variables only to their scope. In C++, declaring a variable is an expensive operation, but the only restriction is that the variable should be declared before it is used [5]. To increase the efficiency, I also declared variables in the minimum scope necessary [5]. Another optimization technique that I used was function overloading in my child classes where I had one function's name set and then repeated the same function name but with different parameters in another class. I also used function overloading of functions outside of my classes as seen in figure 17. Function overloading is considered one of the polymorphism features in C++ because it allows for a multiple of functions to have the same name but with different definitions [7]. This feature was useful for my program when I needed to have the same function run for a map of a different inherited class.

Another programming concept that I used was the Standard Template Library data structure maps and iterators, which were very useful in managing and manipulating my class objects.

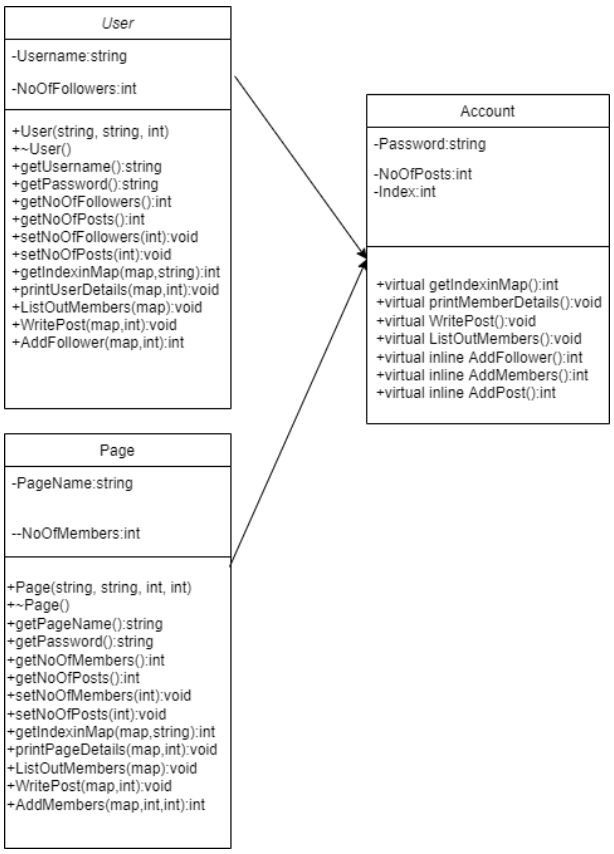

Figure 1: Class structure of social networking platform

```
class Account
```

Figure 2: Class Account

```
class Page : public Account
```

Figure 3: Class Page

```
class User : public Account
```

Figure 4: Class User

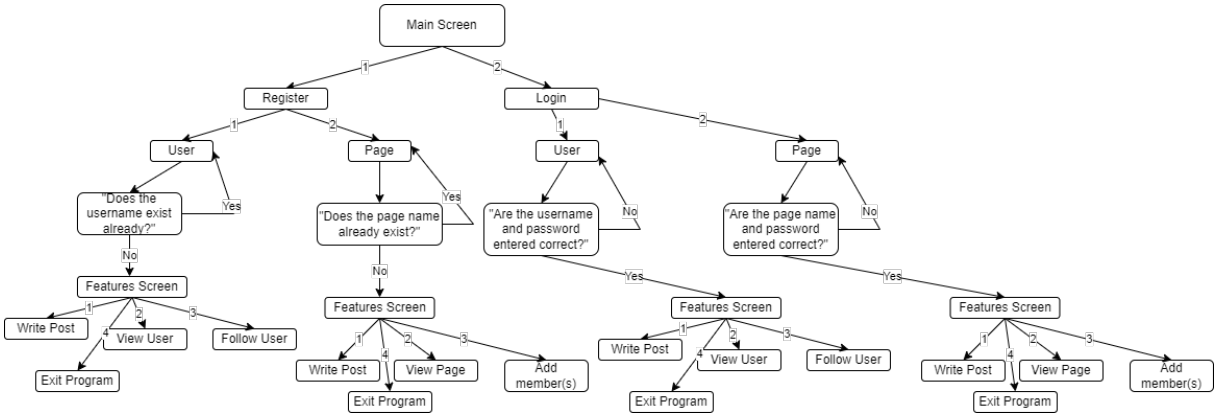

Figure 5: Flowchart diagram of social networking platform

```
//Default Constructor
User()
{
    UserName = " ";
    NoOfFollowers = 0;
    NoOfPosts = 0;
}
//Parametrized Constructor
User(std::string _UserName, std::string _Password, int _NoOfFollowers,
    int _NoOfPosts)
{
    UserName = _UserName;
    Password = _Password;
    NoOfFollowers = _NoOfFollowers;
    NoOfPosts = _NoOfPosts;
}
```

Figure 6: Constructor User

```
//Default constructor
Page() {
    PageName = " ";
    NoOfMembers = 0;
    NoOfPosts = 0;
}
//Parametrized constructor
Page(std::string _PageName, std::string _Password, int _NoOfMembers, int _NoOfPosts)
{
    PageName = _PageName;
    Password = _Password;
    NoOfMembers = _NoOfMembers;
    NoOfPosts = _NoOfPosts;
}
```

Figure 7: Constructor Page

```
//Setters and Getters
std::string getUserName() { return UserName; }
std::string getPassword() { return Password; }
int getNoOfFollowers() { return NoOfFollowers; }
int getNoOfPosts() { return NoOfPosts; }
void setNoOfFollowers(int _NoOfFollowers) { NoOfFollowers = _NoOfFollowers; }
void setNoOfPosts(int _NoOfPosts) { NoOfPosts = _NoOfPosts; }
```

Figure 8: Setters and Getters for User

```
//Setters and getters
std::string getPageName() { return PageName; }
std::string getPassword() { return Password; }
int getNoOfMembers() { return NoOfMembers; }
int getNoOfPosts() { return NoOfPosts; }
void setNoOfMembers(int _NoOfMembers) { NoOfMembers = _NoOfMembers; }
void setNoOfPosts(int _NoOfPosts) { NoOfPosts = _NoOfPosts; }
```

Figure 9: Setters and Getters for Page

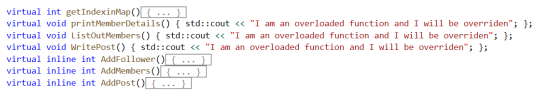

Figure 10: Virtual Functions in the base class

```
//Virtual destructor of class Account
virtual ~Account() {}
```

Figure 11: Virtual Destructor in the base class

```
//Virtual Destructor
virtual ~User() {}
```

Figure 12: Virtual Destructor in the child class

```
//Virtual destructor
virtual ~Page() {}
```

Figure 13: Virtual Destructor in the child class

```
//Function that increments the number of followers when one user follows another user
inline int AddFollower(std::map<int, User> PlaceHolder, int index) {
    //Incrementing the number of followers the user has
    PlaceHolder[index].setNoOfFollowers(PlaceHolder[index].getNoOfFollowers() + 1);
    //Returning the number of followers of a user
    return PlaceHolder[index].getNoOfFollowers();
}
```

Figure 14: Inline function that adds a follower

```
//Function that adds a number of members when one user adds members to their page
inline int AddMembers(std::map<int, Page> PlaceHolder, int index, int number) {
    //Adding the members
    PlaceHolder[index].setNoOfMembers(PlaceHolder[index].getNoOfMembers() + number);
    //returning the number of members of the page
    return PlaceHolder[index].getNoOfMembers();
}
```

Figure 15: Inline function that adds members

```
inline int AddPost(std::map<int, User> PlaceHolder, int index) {
    int result;
    result = PlaceHolder[index].getNoOfPosts() + 1;
    return result;
}
```

Figure 16: Inline function that adds a post

```
//Checking if a user exists using a map iterator and the username and password entered by the user
bool CheckMemberExistence(std::string _UserOrPage, std::string _Password, std::map<int, User> PlaceHolder) { ... }

//Checking if a user exists using a map iterator and the username entered by the user
bool CheckMemberExistence(std::string _UserOrPage, std::map<int, User> PlaceHolder) { ... }

//Checking if a page exists using a map iterator and the page name and password entered by the user
bool CheckMemberExistence(std::string _UserOrPage, std::string _Password, std::map<int, Page> PlaceHolder) { ... }

//Checking if a page exists using a map iterator and the page name and password entered by the user
bool CheckMemberExistence(std::string _UserOrPage, std::map<int, Page> PlaceHolder) { ... }
```

Figure 17: An example of an overloaded function

Bibliography

1. Samur, Alexandra. "The History of Social Media: 29+ Key Moments." Social Media Marketing amp; Management Dashboard, 27 Nov. 2018, https://blog.hootsuite.com/history-social-media/.

2. D.says:, Jim, et al. "How Many People Use Social Media in 2022? (65+ Statistics)." Backlinko, 10 Oct. 2021, https://backlinko.com/social-media-users.

3. Hall, Mark. "Facebook." Encyclopædia Britannica, Encyclopædia Britannica, Inc., https://www.britannica.com/topic/Facebook.

4. Gupta, Rahul. "Virtual Destructor." GeeksforGeeks, 14 Dec. 2021, https://www.geeksforgeeks.org/virtu destructor/.

5. Isensee, Pete. "C++ Optimization Strategies and Techniques." C++ Optimizations You Can Do "As You Go", https://www.tantalon.com/pete/cppopt/asyougo.htmPostponeVariableDeclaration.

6. Pravasi, Meet. "Inline Functions in C++." GeeksforGeeks, 7 Sept. 2018, https://www.geeksforgeeks.org/ functions-cpp/.

7. "Function Overloading in C++." GeeksforGeeks, 17 June 2021, https://www.geeksforgeeks.org/function-overloading-c/.