CSIT 121 - PROJECT

Social Media Platform

Objective

Our project is based on a social media platform that connects users all around the globe and offers a user-friendly experience

Varun Tulsiyani – 8066441

Hawra Virani – 8064027

Raha Habib – 7695391

Table of Contents

Project Proposal

Project Details

Develop a program that manages a social media platform, including user registration, post creation and messaging. The system should allow users to create an account, post content, view other users' content and message other users with a series of simple clicks!

Features

Our main aim is to build an application using object orientated programming in Java that can help users interact by sharing pictures, videos, audios and messaging one another in a captivating setting. This will in turn help users stay in touch in real time. Not only this but this application can be an excellent platform for advertisers who can promote their product by sharing their content and posting advertisements helping them reach a wider audience.

Methodology

To achieve such, we will be focusing on making use of objects, classes, inheritance, polymorphism, encapsulation, abstraction classes and Java graphics which are the pillars of object orientated programming.

Target Audience

Our digital platform is non-target specific and aims to reach a broader audience who find social media to be interesting and would like to utilize it in the best way possible by connecting with new people and wanting to stay in touch with the old ones and even using it as a form of digital marketing media.

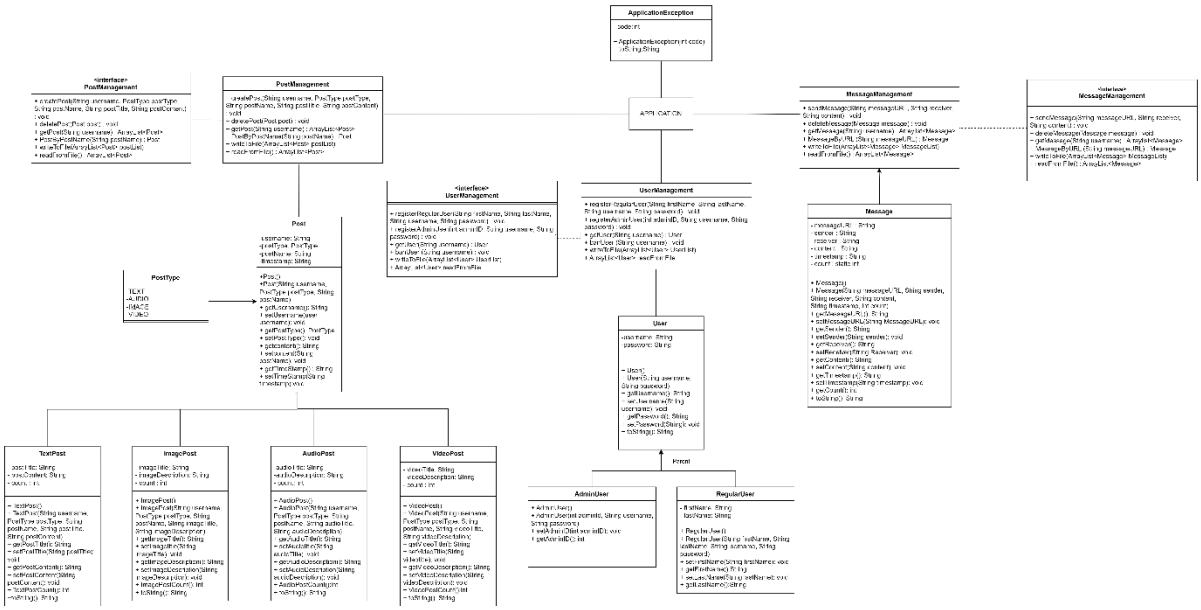
Expected Results

By putting the above into practice, a high functioning application will be put together while being user friendly, accessible, and well-presented.

Conclusion

Our aim is to develop a digital environment that connects individuals, encourages social interaction, and allows for the exchange of content in an exciting manner. We intend to build a dependable and scalable platform that satisfies the requirements of a wide spectrum of social media users utilizing Java and object-oriented programming concepts.

Unified Modeling Language (UML)



Class Design

1. User

This is the parent class of AdminUser and RegularUser. It has two private attributes “username” and “password” of type String. This class has two constructors ‘User()’ as the default constructor and a parameterized constructor ‘User(String username, String password)’. There is also a getter and setter method for both “username” and “password” allowing access and modification of their values. The ‘toString()’ method is included to provide a string representation of the object.

2. AdminUser

This class extends ‘User’ class and inherits its attributes and methods. The default constructor of AdminUser is ‘AdminUser()’ and its parameterized constructor is ‘AdminUser (int adminId, String username, String password)’. The class has a getter method getAdminId() to retrieve the adminId value and a setter method setAdminId(int adminId) to set the adminId value.

3. RegularUser

This class also extends the ‘User’ class, inheriting the attributes and methods of the ‘User’ class. This class has two private attributes called "firstName" and "lastName" of type String. This class has two constructors, a default constructor RegularUser() and a parameterized constructor RegularUser(String firstName, String lastName, String username, String password). There are getter and setter methods in this class for both "firstName" and "lastName" attributes, allowing access and modification of their values.

4. UserManagemant

The class "UserManagement" is responsible for managing instances of the "User" class. It has a private attribute called "users" of type ArrayList<User>, which stores the collection of user objects. The class has the methods:

The registerRegularUser() method takes the first name, last name, username, and password as parameters and registers a regular user.

The registerAdminUser() method takes the admin ID, username, and password as parameters and registers an admin user.

The getUser() method retrieves a user based on the provided username and returns the corresponding User object.

The banUser() method bans a user based on the provided username.

The writeToFile() method takes an ArrayList of User objects as a parameter and writes the user data to a file.

The readFromFile() method reads the user data from a file and returns it as an ArrayList of User objects.

5. Interface (User)

The interface defines the contract or blueprint for managing user-related operations.

6. Post

This is the parent class of TextPost, ImagePost, AudioPost and VideoPost. It has four private attributes “username”, “postType”, “postName” and “timestamp”. This class has two constructors ‘Post()’ as the default constructor and a parameterized constructor ‘Post(String username, PostType postType, String postName)’. There is also a getter and setter method for each attribute.

The "PostType" enumeration represents the predefined set of values for the post type attribute. The enumeration contains four constants: "TEXT," "AUDIO," "IMAGE," and "VIDEO”.

7. TextPost

This class extends ‘Post’ class and inherits its attributes and methods. It has two private attributes: "postTitle" and "postContent" of type String, and "count" of type int. It has a default constructor ‘TextPost()’ and a parameterized constructor ‘TextPost(String username, PostType postType, String postName, String postTitle, String postContent)’. The class has getter and setter methods for the "postTitle" and "postContent" attributes.

The method TextPostCount(): int returns the value of the "count" attribute.

The method toString(): String overrides the default implementation of the toString() method to provide a string representation of the "TextPost" object.

8. ImagePost

This class also extends ‘Post’ class and inherits its attributes and methods. It has two private attributes: "ImageTitle” and "ImageDescription" of type String, and "count" of type int. It has a default constructor ‘ImagePost()’ and a parameterized constructor ‘ImagePost(String username, PostType postType, String postName, String imageTitle, String imageDescription)’. The class has getter and setter methods for the "ImageTitle" and "ImageDescription" attributes.

The method ImagePostCount(): int returns the value of the "count" attribute.

The method toString(): String overrides the default implementation of the toString() method to provide a string representation of the "ImagePost" object.

9. AudioPost

This class also extends ‘Post’ class and inherits its attributes and methods. It has two private attributes: "audioTitle” and "audioDescription" of type String, and "count" of type int. It has a default constructor ‘AudioPost()’ and a parameterized constructor ‘AudioPost(String username, PostType postType, String postName, String audioTitle, String audioDescription)’. The class has getter and setter methods for the "audioTitle" and "audioDescription" attributes. The method TextPostCount(): int returns the value of the "count" attribute.

The method AudioPostCount(): int returns the value of the "count" attribute.

The method toString(): String overrides the default implementation of the toString() method to provide a string representation of the "AudioPost" object.

10. VideoPost

This class also extends ‘Post’ class and inherits its attributes and methods. It has two private attributes: "videoTitle” and "videoDescription" of type String, and "count" of type int. It has a default constructor ‘VideoPost()’ and a parameterized constructor ‘VideoPost(String username, PostType postType, String postName, String videoTitle, String videoDescription)’. The class has getter and setter methods for the "videoTitle" and "videoDescription" attributes. The method TextPostCount(): int returns the value of the "count" attribute.

The method VideoPostCount(): int returns the value of the "count" attribute.

The method toString(): String overrides the default implementation of the toString() method to provide a string representation of the "VideoPost" object.

11. PostManagement

The class "PostManagement" is responsible for managing instances of the "Post" class. It has a private attribute called "posts" of type ArrayList<Post>, which stores the collection of post objects. The class has the methods:

The createPost() method takes the username, post type, post name, post title, and post content as parameters and creates a post.

The deletePost() method takes a Post object as a parameter and deletes it.

The getPosts() method retrieves all the posts for a given username and returns them as an ArrayList of Post objects.

The getPostByPostName() method retrieves a specific post by its post name and returns the corresponding Post object.

The writeToFile() method takes an ArrayList of Post objects as a parameter and writes the post data to a file.

The readFromFile() method reads the post data from a file and returns it as an ArrayList of Post objects.

12. Interface (Post)

This interface provides a contract that classes implementing it must adhere to, ensuring that they provide the specified functionality for managing posts. The implementation details of these methods will be provided in the classes that implement the "PostManagement" interface.

13. Message

The "Message" class has several private attributes: "messageURL", "sender", "receiver", "content" and "timestamp" of type String, and "count" of type static int. This class has two constructors, a default constructor Message() and a parameterized constructor Message(String messageURL, String sender, String receiver, String content, String timestamp, int count). The class has getter and setter methods for all the attributes, allowing access and modification of their values. The toString() method overrides the default implementation to provide a string representation of the "Message" object.

14. MessageManagement

The class "MesageManagement" is responsible for managing instances of the "Message" class. It has a private attribute called "messages" of type ArrayList<Message>, which stores the collection of The "MessageManagement" class provides methods to manage messages.

The sendMessage() method takes the message URL, receiver, and content as parameters and sends the message.

The deleteMessage() method takes a Message object as a parameter and deletes it.

The getMessages() method retrieves all the messages for a given username and returns them as an ArrayList of Message objects.

The getMessageByURL() method retrieves a specific message by its message URL and returns the corresponding Message object.

The writeToFile() method takes an ArrayList of Message objects as a parameter and writes the messages to a file.

The readFromFile() method reads the messages from a file and returns them as an ArrayList of Message objects.

15. Interface (Message)

This interface provides a contract that classes implementing it must adhere to, ensuring that they provide the specified functionality for managing messages. The implementation details of these methods will be provided in the classes that implement the "MessageManagement" interface.

16. ApplicationException

The "ApplicationException" class extends the built-in "Exception" class, indicating that it is an exception class. It has a private attribute "code" of type int, which represents the code associated with the exception. The class has a constructor ApplicationException(int code) that takes the code as a parameter and assigns it to the corresponding attribute. The toString() method is overridden to provide a string representation of the exception, which includes the code.

Classes & Description

Classes	Description
User	Parent class creating objects
AdminUser	Sub class of User creating objects
RegularUser	Sub class of User creating objects
UserManagement	Actions for objects of type User
Post	Parent class creating objects
ImagePost	Sub class of Post creating objects
AudioPost	Sub class of Post creating objects
VideoPost	Sub class of Post creating objects
TextPost	Sub class of Post creating objects
PostManagement	Actions for objects of type Post
Message	Parent class creating objects
MessageManagement	Actions for objects of type Message
Interface	Initialized methods overridden by classes
Exception	Utilized for all classes
Driver	Testing the entire program

Detailed Description of Classes

1. User: One of the main classes of our platform. It expresses the essential features of the common attributes of all users in the system.
2. AdminUser: This is the sub-class of the main class User. It extends the functionality of the parent class and contains admin users that would be regulating the platform.
3. RegularUser: This is another sub-class of User. And allows new user and existing user to access the online platform.
4. UserManagement: This is the class that overlooks the management of the users on the platform.
5. Post: The second main class that governs the different types of posts.
6. ImagePost: Sub-class of Post that allows the user to post an image.
7. AudioPost: Sub-class of Post that allows the user to post an audio.
8. VideoPost: Sub-class of Post that allows the user to post a video.
9. TextPost: Sub-class of Post that allows the user to post a text.
10. PostManagement: This is the class that overlooks and manages the posting of the users on the platform.
11. Message: The final main class of the platform that governs the messages that the users input.
12. MessageManagement: This is the class that overlooks and manages the messaging of the users.
13. Interface: A separate class that contains common methods of the management classes.
14. Exception: Another separate class that deals with the exceptions that are common/accessible by all the classes.
15. Driver: The main and final class that tests all the existing classes.

Application Explanation & Screenshots

Below are the screenshots of the important classes and their respective features.

Create Account Feature:

```
14      public void registerRegularUser(String firstName, String lastName, String username, String password) {
15          userList = readFromFile();
16          RegularUser newUser = new RegularUser(firstName, lastName, username, password);
17          userList.add(newUser);
18          writeToFile(userList);
19      }
```

registerRegularUser is a creation method that allows the users to be registered on the platform and hence can create an account on the platform.

Similar Creation Methods:

1. registerAdminUser (UserManagement Class)
2. createPost (PostManagement Class)
3. createMessage (MessageManagement Class)

Get an Object Feature:

```
38      public ArrayList<Message> getMessage(String username) {
39          ArrayList<Message> userMessages = new ArrayList<>();
40          messageList = readFromFile();
41
42          for (Message m : messageList) {
43              if (m.getSender().equals(username)) {
44                  userMessages.add(m);
45              }
46          }
47          return userMessages;
48      }
```

This method is used to get an object instance from an array list.

Similar Get Methods:

1. getUser (UserManagement)
2. getPost (PostManagement)

Delete Method:

```java
public void banUser(int adminID, String username) {
    userList = readFromFile();
    User userToRemove = null;

    for (User user : userList) {
        if (user.getUsername().equals(username)) {
            userToRemove = user;
            break;
        }
    }

    if (userToRemove != null) {
        userList.remove(userToRemove);
        writeToFile(userList);
        System.out.println(x:"User banned successfully!");
    } else {
        System.out.println(x:"User not found!");
    }
}
```

This method allows the admin user to delete/ban a user completely. Doing so, helps in the regulation and monitoring of the platform.

Similar Delete Methods:

1. Delete Post (PostManagement Class)
2. Delete Message (MessageManagement Class)

Writing ArrayList Objects into Files:

```java
public void writeToFile(ArrayList<User> userList) {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
        for (User user : userList) {
            if (user instanceof RegularUser) {
                RegularUser regularUser = (RegularUser) user;
                writer.write(regularUser.getFirstName() + "," + regularUser.getLastName() + ","
                        + regularUser.getUsername() + "," + regularUser.getPassword() + ",Regular");
            } else if (user instanceof AdminUser) {
                AdminUser adminUser = (AdminUser) user;
                writer.write(adminUser.getAdminID() + "," + adminUser.getUsername() + ","
                        + adminUser.getPassword() + ",Admin");
            }
            writer.newLine();
        }
    } catch (IOException e) {
        System.out.println("Error writing to users file: " + e.getMessage());
    }
}
```

The method writeToFile() is defined uniquely in different classes depending on the requirements.

Similar File Methods: Read from File

Read from File:

```
78      public ArrayList<User> readFromFile() {
79          ArrayList<User> readUserList = new ArrayList<>();
80
81          try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
82              String line;
83              while ((line = reader.readLine()) != null) {
84                  String[] accountData = line.split(regex:",");
85                  if (accountData.length == 4) {
86                      int savedAdminID = Integer.parseInt(accountData[0]);
87                      String savedUsername = accountData[1],
88                             savedPassword = accountData[2];
89
90                      readUserList.add(new AdminUser(savedAdminID, savedUsername, savedPassword));
91                  } else if (accountData.length == 5) {
92                      String savedFirstName = accountData[0],
93                             savedLastName = accountData[1],
94                             savedUsername = accountData[2],
95                             savedPassword = accountData[3];
96
97                      readUserList.add(new RegularUser(savedFirstName, savedLastName, savedUsername,
                        savedPassword));
98                  }
99              }
100         } catch (IOException e) {
101             System.out.println("Error reading users file: " + e.getMessage());
102         }
103         return readUserList;
104     }
```

Interface:

1. *UserMgmt:*

```
1   package project;
2
3   import java.util.ArrayList;
4
5   public interface InterfaceUserMgmt {
6       void registerRegularUser(String firstName, String lastName, String username, String password);
7
8       void registerAdminUser(int adminID, String username, String password);
9
10      User getUser(String username);
11
12      void banUser(int adminID, String username);
13
14      void writeToFile(ArrayList<User> userList);
15
16      ArrayList<User> readFromFile();
17  }
```

2. *PostMgmt:*

```java
package project;

import java.util.ArrayList;

public interface InterfacePostMgmt {
    void createPost(String username, PostType postType, String postName, String postTitle, String
    postContent);

    void deletePost(Post post);

    ArrayList<Post> getPost(String username);

    Post PostByPostName(String postName);

    void writeToFile(ArrayList<Post> postList);

    ArrayList<Post> readFromFile();
}
```

3. *MessageMgmt*

```java
package project;

import java.util.ArrayList;

public interface InterfaceMessageMgmt {
    void sendMessage(String messageURL, String sender, String receiver, String content);

    void deleteMessage(Message message);

    ArrayList<Message> getMessage(String username);

    Message MessageByURL(String messageURL);

    void writeToFile(ArrayList<Message> messageList);

    ArrayList<Message> readFromFile();
}
```

An interface has been implemented in all the management classes that has similar methods for each, overridden in each respective class.

Exception Class:

```java
package project;

public class ApplicationException extends Exception {
    private int code;

    ApplicationException(int code) {
        this.code = code;
    }

    public String toString() {
        String message = "";
        if (code == 1) {
            message = "Invalid choice!";
        } else if (code == 2) {
            message = "Invalid username or password!";
        } else if (code == 3) {
            message = "Username already exists!";
        }

        return message;
    }
}
```

An exception class has been implemented to deal with the following errors: -

- If the user has selected an invalid/choice that doesn't exist apart from the given list, it shows an "Invalid choice!" exception.
- If the user has entered the incorrect credentials either the username or the password upon logging in, it shows an "Invalid username or password!" exception.
- And lastly, if the user is trying to pick a username upon signing up that's already taken by another user, it displays an "Username already exists!" error.

Driver Class:

Out of the box feature:

1. **Enum**

```
14      private static final int LOGIN = 1, SIGNUP = 2, EXIT = 3;
15      private static final int REGULAR_USER = 1, ADMIN_USER = 2;
16      private static final int TEXT = 1, AUDIO = 2, IMAGE = 3, VIDEO = 4;
17      private static final int POST = 1, MESSAGE = 2;
18      private static final int DELETE_POST = 1, GET_USER = 2, BAN_USER = 3, SIGNOUT = 4;
19      private static final int ADD = 1, DELETE = 2, GET = 3;
```

This block of code sets a value to a word and finalizes it using the final keyword. These words can be used as a replacement of the value assigned them to understand the code better.

2. **ClearScreen**

```
338     public static void ClearScreen() {
339         System.out.println(x:"\033[H\033[2J");
340         System.out.flush();
341     }
```

This method clears the console after the method is called.

3. **TimeStamp**

```
24      public Post(String username, PostType postType, String postName) {
25          this.username = username;
26          this.postType = postType;
27          this.postName = postName;
28          timestamp = new SimpleDateFormat(pattern:"yyyy.MM.dd HH.mm.ss").format(new Date());
29      }
30
```

This syntax provides the current time in the given pattern/format for features like posting and sending/receiving messages.

Additional Features:

Login

```
55      public static void login() throws ApplicationException {
56          ClearScreen();
57
58          System.out.println(x:"---Log In---");
59          System.out.print(s:"Username: ");
60          username = in.next();
61          System.out.print(s:"Password: ");
62          password = in.next();
63
64          ArrayList<User> userList = um.readFromFile();
65          for (User user : userList) {
66              if (user.getUsername().equals(username) && user.getPassword().equals(password)) {
67                  currentUser = username;
68                  break;
69              }
70          }
71      }
```

This method allows the pre-registered user to log in to the platform after entering their respective credentials. If the user accidentally enters the incorrect username, it throws an exception and gives the user an error.

Signup

```
88  public static void signup() throws ApplicationException {
89      ClearScreen();
90
91      System.out.println(x:"---Sign Up---");
92      System.out.println("1. Regular User" + "\n2. Admin User");
93      choice1 = in.nextInt();
94
95      System.out.print(s:"Username: ");
96      username = in.next();
97      ArrayList<User> userList = um.readFromFile();
98      for (User user : userList) {
99          if (user.getUsername().equals(username)) {
100             throw new ApplicationException(code:3);
101         }
102     }
103
104     System.out.print(s:"Password: ");
105     password = in.next();
106
107     if (choice1 == REGULAR_USER) {
108         System.out.print(s:"First Name: ");
109         firstName = in.next();
110         System.out.print(s:"Last Name: ");
111         lastName = in.next();
112         um.registerRegularUser(firstName, lastName, username, password);
113     } else if (choice1 == ADMIN_USER) {
114         System.out.print(s:"Admin ID: ");
115         adminID = in.nextInt();
116         um.registerAdminUser(adminID, username, password);
117     }
118     System.out.println(x:"Sign Up successful");
119     login();
120 }
```

This block of code helps in a user signing up, it first clears the screen to remove any pre-existing choice window. Then gives the user a choice to sign and gives them 2 options to choose from

a- Admin user
b- Regular user

It then asks the user to enter their username upon which the system then matches the username from the file existing in the database whether it exists. If the username is already taken it throws an exception and asks the user to choose a different username since the primary key or a unique identifier of this platform is the unique username of every individual.

It then prompts the user to enter their password,

And the signup is successful.

Whereas for the AdminUser it asks for the adminID it prompts for the username and password and allows them to sign up.

Add Method

```
107     public static void add() throws ApplicationException {
108         ClearScreen();
109 
110         System.out.println("1. Post" + "\n2. Message");
111         choice1 = in.nextInt();
112 
113         if (choice1 == POST) {
114             System.out.println(x:"---ADD POST---");
115             System.out.println("1. Text Post" + "\n2. Audio Post" + "\n3. Image Post" + "\n4. Video Post");
116             choice1 = in.nextInt();
117             switch (choice1) {
118                 case TEXT:
119                     System.out.println(x:"---TEXT---");
120                     postType = PostType.TEXT;
121                     postName = currentUser + "text" + TextPost.TextPostCount();
122 
123                     System.out.println(x:"Text Title: ");
124                     postTitle = in.next();
125                     System.out.println(x:"Text Content: ");
126                     postContent = in.next();
127 
128                     pm.createPost(currentUser, postType, postName, postTitle, postContent);
129                     break;
```

Add method allows the user to choose from either adding a post or a message:

Add Post has several options to choose from such as:

- Text Post
- Audio Post
- Image Post
- Video Post

And then allows them to put a text to their content, title etc.

```
169         } else if (choice1 == MESSAGE) {
170             System.out.println(x:"---MESSAGE---");
171             String messageURL = currentUser + Message.getCount();
172 
173             System.out.println(x:"Receiver: ");
174             String receiver = in.next();
175             System.out.println(x:"Message: ");
176             String content = in.next();
177 
178             mm.sendMessage(messageURL, currentUser, receiver, content);
179         } else {
180             throw new ApplicationException(INVALID_CHOICE);
181         }
```

Similarly, it allows the user to add a message by mentioning the receiver and the inputting the message in the String format.

The message is then displayed in the following format-

messageURL, currentUser, receiver and content.

Delete Method

```
189     public static void delete() throws ApplicationException {
190         ClearScreen();
191 
192         System.out.println("1. Post" + "\n2. Message");
193         choice1 = in.nextInt();
194 
195         if (choice1 == POST) {
196             System.out.println(x:"---POST---");
197             System.out.print(s:"Post Name: ");
198             String postName = in.next();
199             pm.deletePost(pm.PostByPostName(postName));
200         } else if (choice1 == MESSAGE) {
201             System.out.println(x:"---MESSAGE---");
202             System.out.println(x:"Message URL: ");
203             String messageURL = in.next();
204             mm.deleteMessage(mm.MessageByURL(messageURL));
205         } else {
206             throw new ApplicationException(INVALID_CHOICE);
207         }
208     }
```

Delete method allows either the post or the message to be deleted. The post is identified by the Post Name and the message is identified by its respective URL.

Get Method

```
210     public static void get() throws ApplicationException {
211         ClearScreen();
212 
213         System.out.println("1. Post" + "\n2. Message");
214         choice1 = in.nextInt();
215 
216         if (choice1 == POST) {
217             System.out.println(x:"---POST---");
218             System.out.println(pm.getPost(currentUser));
219         } else if (choice1 == MESSAGE) {
220             System.out.println(x:"---MESSAGE---");
221             System.out.println(mm.getMessage(currentUser));
222         } else {
223             throw new ApplicationException(INVALID_CHOICE);
224 
225         }
226     }
```

This method allows the user to get/display either their post or message.

Distribution of Tasks

Team Member	Task Implemented
Varun Tulsiyani	• Management classes • Exception • Read and write files • Driver
Hawra Virani	• UML • Sub classes • Interface • Report Work
Raha Habib	• Project proposal • Class Design • Parent classes • Report work