Workshop 2

ANDRES CAMILO RAMOS ROJAS – 20242020005

Project:

Outlook (message app with a close environment)

Objectives:

- The user will registrant his self with an identificatory
- The user will see all the other users that exist in the system
- The user will be able to send messages to other users

Requirements:

- Functional requirements:
  - User register
  - User login
  - User identification
  - User communication
  - User update (was added with human error in mind)
  - User elimination (was added for users that leave the system)
  - User logout
  - Message management
  - Archive management
- Non - functional requirements:
  - Multiple users simultaneously support
  - Outlook-like interface
  - Standard security parameters to password defining
  - Unique identifiers (usernames)

User stories:

Title: user register	Priority: high	Estimate:
User story: As a user , I want to register my self, so that I can send messages to other users.		
Acceptance criteria: Given a user When he register her self Then he will be able to sing in		

Title: user access	Priority: High	Estimate
User story:		

As a user, I want to be able to sing in, so that I will have acces to the system.
Acceptance criteria: Given 10 users When they acces to the system Then will be able to send a message

Title: user identification	Priority: high	Estimate:
User story: As a user, I want have an identifier , so that my familiars, acquaintances and another users recognize me on the system.		
Acceptance criteria: Given hundreds of users When they belong to the system Then they will be able yo recognizes each other easily		

Title: user communication	Priority: high	Estimate;
User story: As a user, I want to send messages to the other users when I need to, so that I can communicate with other people.		
Acceptance criteria: Given hundreds users When they send a message to other user Then the second user will have the message in her messages		

Title: user update	Priority: high	Estimate;
User story: As a user, I want to update my profile, so that If I make a mistake when registering I can correct it.		
Acceptance criteria: Given a user When he update his information Then he will se the changes		

(If a user want to change his password for security reasons, also if there was an error in the register)

Title: user disable	Priority: high	Estimate;
User story: As an administrator, I want to disable profiles, so that I can manage how many users the system have		
Acceptance criteria:		

Title: user log out	Priority: High	Estimate
User story:		

As a user, I want to be able to log out, so that I will disconnect from the system
Acceptance criteria: Given 10 users When they log out Then they will be able to access again

(If there is a user without use the administrator will be able to eliminate this user and it wont be able to login and receive messages)

Title: message query	Priority: high	Estimate:
User story: As a user, I want to see the messages the people send me, so that I can stay in contact with them.		
Acceptance criteria: Given hundred users When the user receive a message Then the user will be able to see al the messages he received		

Title: chats differentiation	Priority: high	Estimate:
User story: As a user, I want a menu of important chats , so that I can I can differentiate between important chats and those that are not		
Acceptance criteria: Given 10 chats When the user mark a chat as important Then he will be able to see faster the more important messages		

Title: archive management	Priority: low	Estimate
User story: As a user, I want to be able to attach archives, so that I can send photos or reports to other people.		
Acceptance criteria: Given a chat When the user attach an archive Then the other user will see and be able to open the archive		

Title:	Priority:	Estimate
User story: As a user, I want to change the color of the interface , so that if I'm used to dark interfaces		
Acceptance criteria: Given [how things begin] When [action taken]		

Then [outcome of taking action]

Mockups:

https://www.figma.com/design/MdKn3vvMIK9hrLO47QKTft/Untitled?node-id=2-127&t=jVvi57LOTBsAzuqb-1

CRC cards:

Class: user	
Responsibility: ○ Register ○ Sing in ○ Log out ○ Update ○ elimination ○ Send messages ○ Read messages	Collaborator: ○ Message ○ Chat

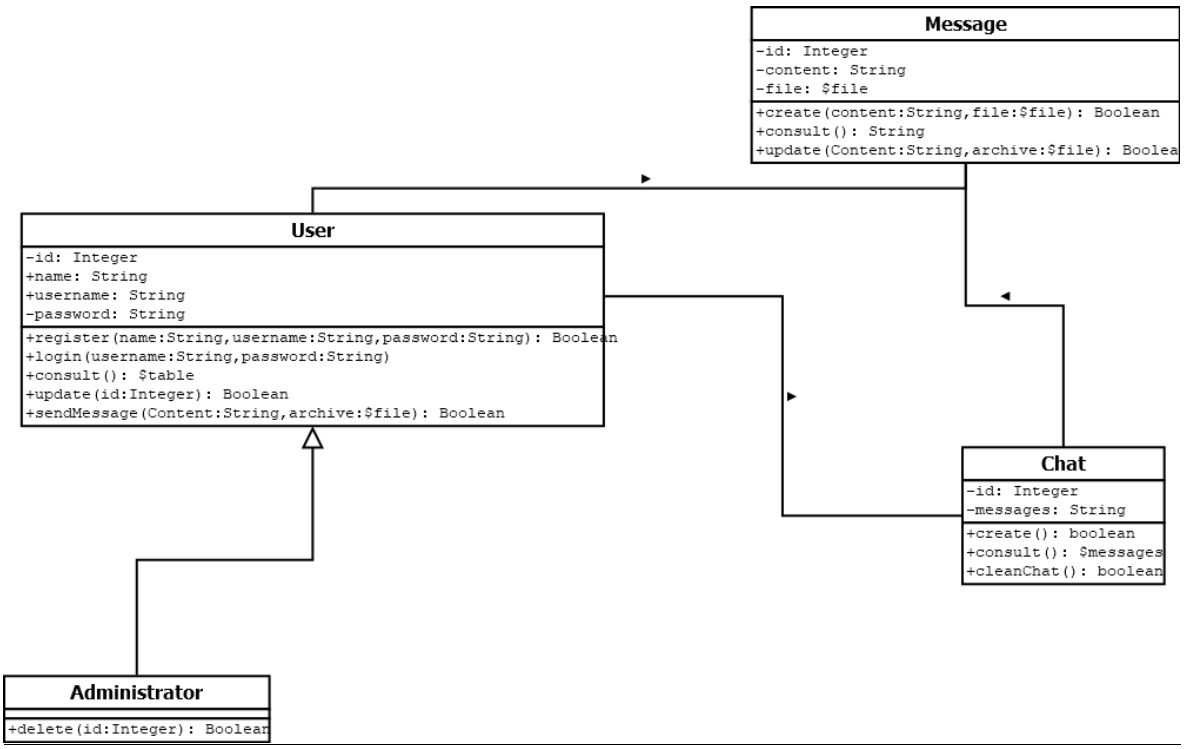
Class: Message	
Responsibility: ○ Create ○ Consult ○ update	Collaborator: ○ User ○ Chat

Class: Chat	
Responsibility: ○ Create ○ Consult ○ Clean chat	Collaborator: ○ User ○ Chat

All the crc card have changed to be equal to the classes diagram, the responsibilities as the methods of each class

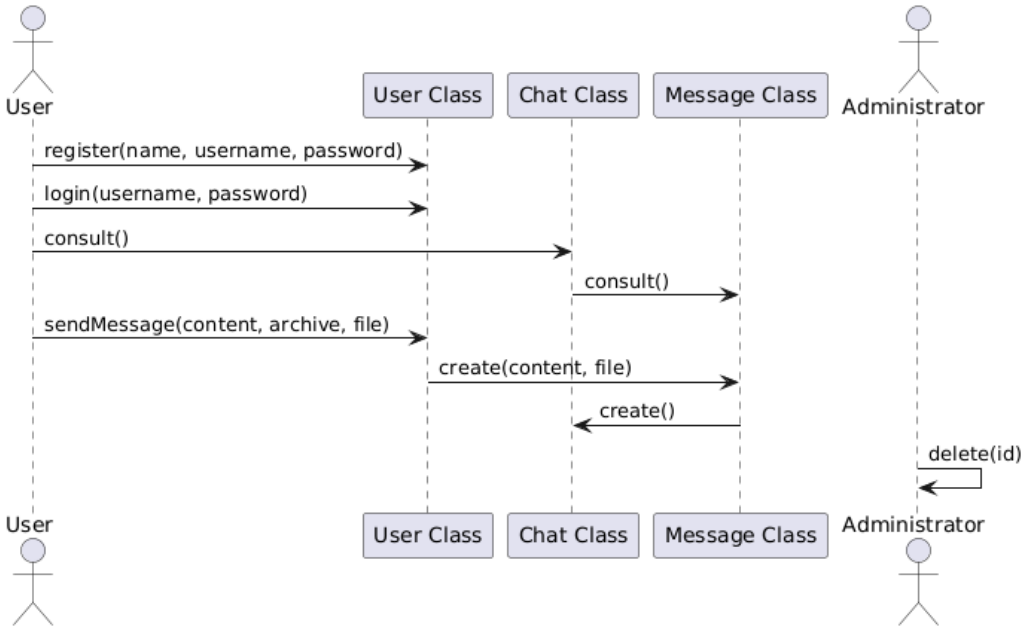
PART 2 – WORKSHOP – 2

UML CLASSES DIAGRAM



The class user has method to satisfy the sing up, sing in, user update, and elimination requirements, the user identification will be able by the attribute username in the user class, the message and archive management is between the classes message and chat, where we will be able to store and show the messages. The same with the archive management

Sequence diagram



IMPLEMENTATION PLAN FOR OOP CONCEPTS

In the project we can see the inheritance working because the super class user has a sub-class administrator that have more methods, the classes message and chat have private attributes, I'll apply the encapsulation by using a validations of the user and if the user is in the chat, he can see it, he also will have permission to send new messages.

The project will use MVC structure, starting with the models proposed in database, the views and all the controllers that will procesate the petitions from the user

Models

- User model
- Chat model
- Message model

Views

- Administrator views
    - Elimination views
    - Chat view
    - user view
    - general view
- User views
    - Chat view
    - user view
    - general view

controllers

- user validation
- validation of information for update

CODE PROGRESS

CLASS USER

```
public class User {
    private Integer id;
    public String name;
    public String username;
    private String password;

    public User (Integer id, String name, String username, String password)
{
        this.id = id;
        this.name = name;
        this.username = username;
        this.password = password;
    }
    public boolean register(String name, String username, String password) {
        /*
        * This method will save the information of a user in the database.
        *
        * @param name: the name of the user
        * @param username: the username of the user
        * @param password: the password of the user
        *
        * @return a confirmation of the registration
        */
        this.name = name;
        this.username = username;
        this.password = password;
        return true;
    }
    public boolean login(String username, String password) {
        /*
        * This confirms the user exists, and if the credentials ingresed are
correct.
        *
        * @param username: the username of the user
        * @param password: the password of the user
        *
        * @return a confirmation of the login, in case the information is
correct, otherwise false
        */
        if (this.username.equals(username) &&
this.password.equals(password)) {
            return true;
```

```
        } else {
            return false;
        }
    }
    public String consult() {
        /*
        * this method should show the information of all the users in the
system, their name an their username
        *
        * @return all the users registered in the system, their name an
their username
        */

        //this method should validate if the user is logged in before
returning the information, an also show the information of all the users in
the system
        return "Name: " + this.name + "\nUsername: " + this.username +
"\nPassword: " + this.password;
    }
    public boolean update(Integer id, String newName, String newUsername,
String newPassword) {
        /*
        * This method will update the information of a user in the database.
        *
        * @param id: the id of the user
        * @param newName: the name of the user
        * @param newUsername: the username of the user
        * @param newPassword: the password of the user
        *
        * @return a confirmation of the update
        */
        //the id will be used to identify the user in the system in the
database
        this.name = newName;
        this.username = newUsername;
        this.password = newPassword;
        return true;
    }
    public boolean sendMessage(String message) {
        /*
        * This method will save the content of a message in the database.
        *
        * @param message: the message to be sent
        *
        * @return a confirmation of the message sent
```

```
        */
        //this method should validate if the user is logged in before
sending the message, and take the message from the interface and send it to
the database in asociation with a chat
        return true;
    }
    public boolean logout() {
        /*
        * This method will log out the user from the system.
        *
        * @return a confirmation of the logout
        */
        this.username = null;
        this.password = null;
        return true;
    }

}
```

CLASS MESSAGE

```
public class Message {
        private Integer id;
    private String content;
    private File file;//the message is suposed to support files.

    public boolean create(String content, File file) {
        /*
         * This method will create a new message in the database.
         *
         * @param content: the content of the message
         * @param file: the file attached to the message
         *
         * @return a confirmation of the creation
         */
        this.content = content;
        this.file = file;
        return true;
    }
    public boolean consult() {
        /*
         * This method will show the content of a message.
         *
         * @return the content of the message
         */
```

```
        //this method should validate if the user is logged in before
returning the information, an also show the information of all the messages
in the system
        return this.content;
    }
    public boolean update(Integer id, String newContent, File newFile) {
        /*
         * This method will update the content of a message in the database.
         *
         * @param id: the id of the message
         * @param newContent: the new content of the message
         * @param newFile: the new file attached to the message
         *
         * @return a confirmation of the update
         */
        //the id will be used to identify the message in the system in the
database
        this.content = newContent;
        this.file = newFile;
        return true;
    }
}
```

CLASS CHAT

```
public class Chat {
    private Integer id;
    private String messages;

    public boolean create(String messages) {
        /*
         * This method will create a new chat in the database.
         *
         * @param messages: the messages of the chat
         *
         * @return a confirmation of the creation
         */
        this.messages = messages;
        return true;
    }
    public String consult() {
        /*
         * This method will show the messages of a chat.
         *
```

```
         * @return the messages of the chat
         */
        //this method should validate if the user is logged in before
returning the information, an also show the information of all the messages
in the system
        return this.messages;
    }
    public boolean cleanChat(Integer id) {
        /*
         * This method will delete a chat in the database.
         *
         * @param id: the id of the chat
         *
         * @return a confirmation of the deletion
         */
        //this method should validate the id of the chat in the system in
the database, and if the user is logged in before cleaning the chat
        this.messages = null;
        return true;
    }
}
```