Workshop 3

ANDRES CAMILO RAMOS ROJAS – 20242020005

**Project**:

Outlook ( message app with a close environment)

this is a school project done in order to learn and apply concepts of object-oriented programming, it was decided to create a replica of an Outlook-style messaging service through object-oriented programming as is the objective of the course, additionally it is expected to achieve a graphical interface.

**Objectives:**

* Create the design for a system that will allow the users to communicate between them.
* Create a prototype of the system planned, and start verifying if there are any design error to correct.
* Create the final product with all the corrections.

**Requirements:**

* Functional requirements:
  + User register:

The user will be able to register himself

* + User login

The user will be able to login in his account

* + User identification

The user will be able to identify other users by the username

* + User communication

The user will be able to send messages to other users

* + User update (was added with human error in mind WS#2)

The user will be able to update his own information

* + User elimination (was added for users that leave the system WS#2)

The administrator will be able to eliminate users

* + User logout

The user will be able to logout from his account

* + Message management

The user will be able to view messages sended by other users

* + Archive management

The user will be able to manage the send and reception of archives

* 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 WS#2)

|  |  |  |
| --- | --- | --- |
| 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 WS#2)

|  |  |  |
| --- | --- | --- |
| 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 |

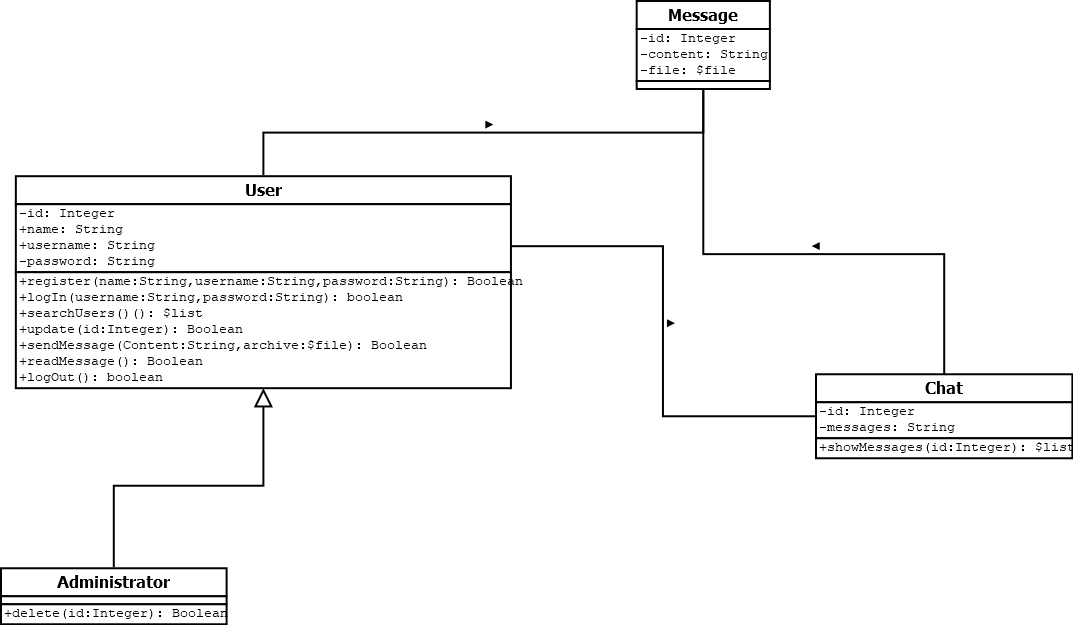
|  |  |
| --- | --- |
| Class: Message | |
| Responsibility: | Collaborator:   * User * Chat |

|  |  |
| --- | --- |
| Class: Chat | |
| Responsibility:   * showMessages | Collaborator:   * User * Chat |

All the crc card have changed to be equal to the classes diagram, the responsibilities as the methods of each class WS#2

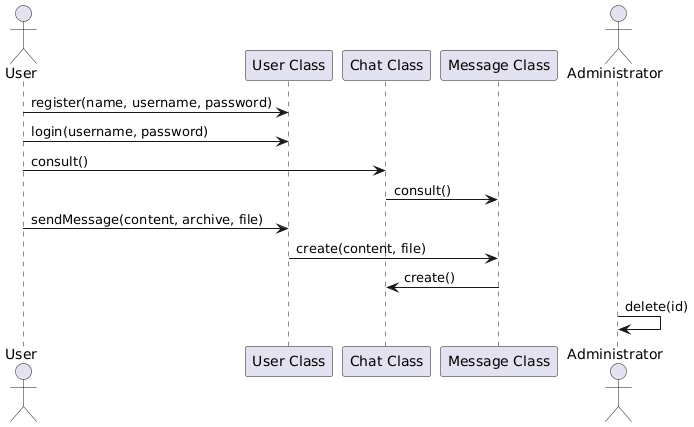
**PART 2 – WORKSHOP – 2**

**UML CLASSES DIAGRAM**

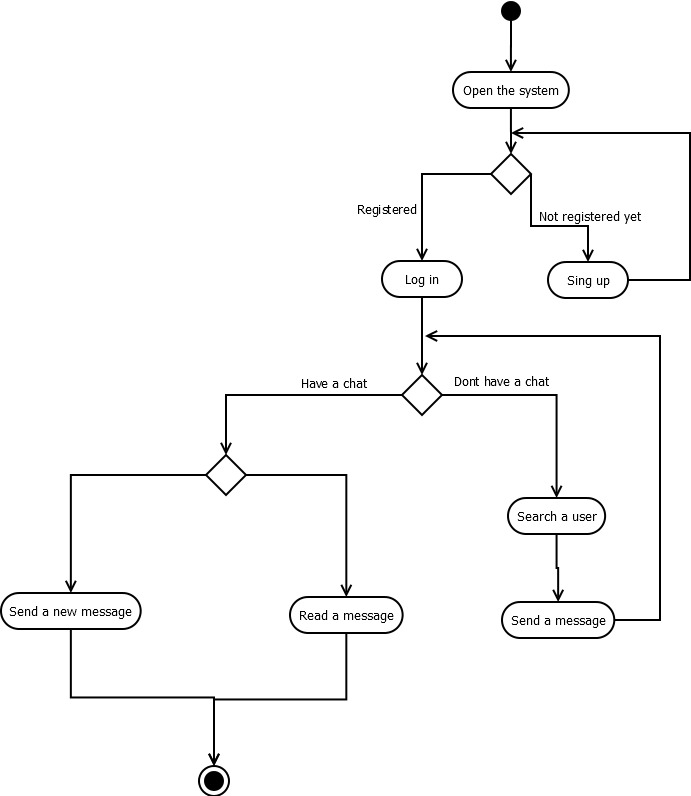


The class user has method to satisfy the sing up, sing in, log out, user update, and elimination requirements, the user identification will be able by the attribute username in the user class, the message and archive management is between the classes message and chat, where we will be able to store and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

**Sequence diagram**



**Activities 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 searchUsers() {

*/\**

*\* 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 readMessage() {

*/\**

*\* This method will read the content of the messages the user has received.*

*\**

*\* @return the content of the message*

*\*/*

*//this method should validate if the user is logged in before reading 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.*

}

**CLASS CHAT**

public class Chat {

    private Integer **id**;

    private String **messages**;

    public boolean showMessages(Integer Id;) {

*/\**

*\* This method will show the messages in the chat.*

*\**

*\* @param id: the identifier of the chat*

*\**

*\* @return mesagges in the chat*

*\*/*

        this.messages = messages;

        return true;*//this should return the messages in the chat, but for now it will return true.*

    }

}

**PART 3 – WORKSHOP – 3**

* **Revisiting Requirements & Design:**
* Neither the requirements have changes nor the CRC cards
* **Enhanced UML Diagrams:**

Diagrama

El contenido generado por IA puede ser incorrecto.

**The class user have changed: Now there is an interface called user that will allow the liskov’s substitution.**

**The relationships have changed: now the user isn’t related to message and the aggregation an composition are now applied to the system.**

**Now these principles allow a better transaction flow in the system because is not a circular cycle and this means that the system will be more efficient and organized.**

* **SOLID-Focused Implementation:**
  + **Single responsibility:**

**Is added by splitting the classes in 3 main classes, user, chat and message, also with the organization, in the update for the workshop 3 the classes message and chat have an unique responsibility being related in line, not in a “circle” like in the past version.**

* + **Open/Closed:**

**In the case I need a new user type for the program I can add a new class from the interface so is open to the extension but closed to the modifications**

* + **Liskov substitution:**

**The interface allows me to do the liskov’s substitution, using code that was thinked for any user in general for a specific type of user without corrupting the code or generating an error**

* + **Interface segregation:**

**The classes message and chat are not forced to implement methods that are not needed, only the user interface has methods in booth classes.**

* + **Dependency inversion:**

**For this principle I can reference the interface user to make the code general an usable for the classes user an administrator**

* **Work in porgress Code & Documentation:**