Object-Oriented Programming

PROJECT
SKYPE

Kevin Navarrete
Valeria Usma

CONTENT

OBJECTIVES

Make an app where the users can send and receive messages, make voice and video calls and share activities with their partners.

Implement a system or an authenticator where every user can have a personal account.

Design a website that is intuitive and easy to use for everyone, with essential functionalities and a seamless user experience that enhances accessibility, responsiveness, and user engagement

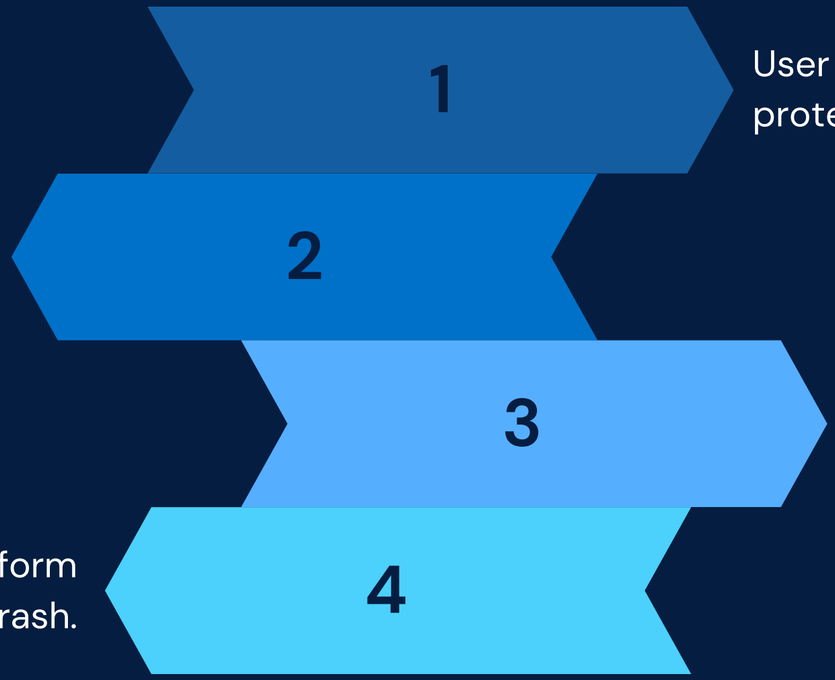
REQUIREMENTS

FUNCTIONAL

requirements

1. Register and log in with email and password.
2. Send and receive private messages to other users.
3. Make voice calls.
4. Public and private group creation and management.
5. Contact handling and user profile customization.
6. Account control (temporary suspension, deletion).
7. Invitation system for friends and groups.

NON FUNCTIONAL
requirements



1. User data and communications must be protected.
2. The system prevents duplicate usernames and email addresses when creating an account.
3. Account changes must be saved and applied instantly.
4. Multiple users can use the platform without causing it to crash.

USER STORIES

Tittle	Priority high	

User Storie:

As a new user, I want to be able to use my email to create a personal account in the app with a unique username and password.

Aceptance Criteria:

1. The user must enter a valid email, unique username, and password to complete registration.
2. If the email or username is already in use, the system displays an appropriate error message.
3. Upon successful registration, the user is automatically logged in and redirected to the main interface.

Tittle	Priority high	

User Storie:

As a user, I want to be able to communicate with other users through messages and calls while maintaining private conversations.

Aceptance Criteria:

1. Users can send and receive messages in a private chat with another user.
2. Messages and calls are encrypted and accessible only by the participants.
3. Users can start voice or video calls directly from the chat interface.

Ttitle	Priority high	

User Storie:

As a user, I want to be able to create or delete groups in the app.

Aceptance Criteria:

1. The user can access a "Create Group" and "Delete Group" option from the group settings.
2. Only group owners or admins can delete a group.
3. Deleting a group removes all chat history for all members.

Tittle	Priority high	

User Storie:

As a user, I want to send messages in a group or chat and have other people see them instantly and be able to reply to them.

Aceptance Criteria:

1. Messages sent in group or private chats appear instantly to all participants.
2. Each message shows the sender's name and timestamp.
3. Participants can reply by sending a new message in the same thread.

Tittle	Priority high	

User Storie:

As a user, I want to be able to add or remove other users from a group I own.

Aceptance Criteria:

1. Group owners can search for users and add them to the group.
2. Group owners can remove any member from the group.
3. A confirmation prompt appears before removing a user.

Tittle	Priority mid	

User Storie:

As a user, I want to temporarily close my account securely to protect it while I'm not using the app.

Aceptance Criteria:

1. The user can deactivate their account from settings.
2. While deactivated, the account is hidden from other users and cannot be used.
3. The user can reactivate the account by logging in again.

Tittle	Priority	
	mid	

User Storie:

As a user, I want to create public or private groups to chat with other users.

Aceptance Criteria:

1. The user can create a group and choose whether it’s public or private.
2. Public groups are visible to all users; private groups require an invitation.
3. The creator becomes the default group admin.

Tittle	Priority	
	mid	

User Storie:

As a user, I want to be able to join existing groups to participate in new conversations.

Aceptance Criteria:

1. Users can view and search for public groups.
2. Users can request to join private groups or accept an invitation.
3. Once joined, the user can read and send messages in the group.

Tittle	Priority	
	mid	

User Storie:

As a user, I want my groups to be manageable only by me or other users to whom I give permission.

Aceptance Criteria:

1. The group owner can assign admin roles to specific members.
2. Only the owner and admins can manage group settings and members.
3. Admin rights can be revoked at any time by the group owner.

Tittle	Priority	
	low	

User Storie:

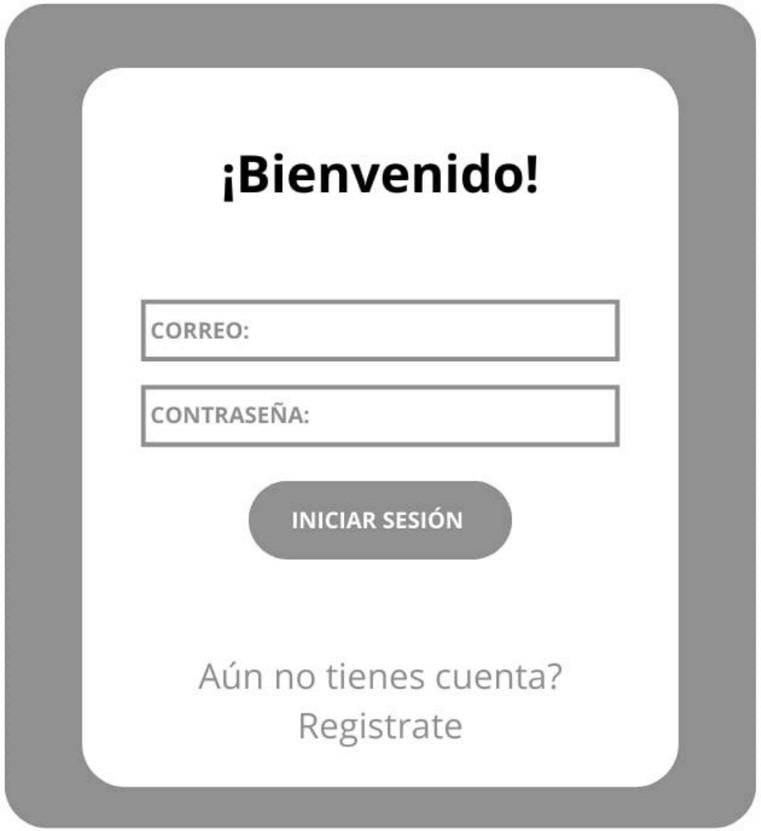
As a user, I want a user-friendly and customizable interface so I can adjust it to my liking.

Aceptance Criteria:

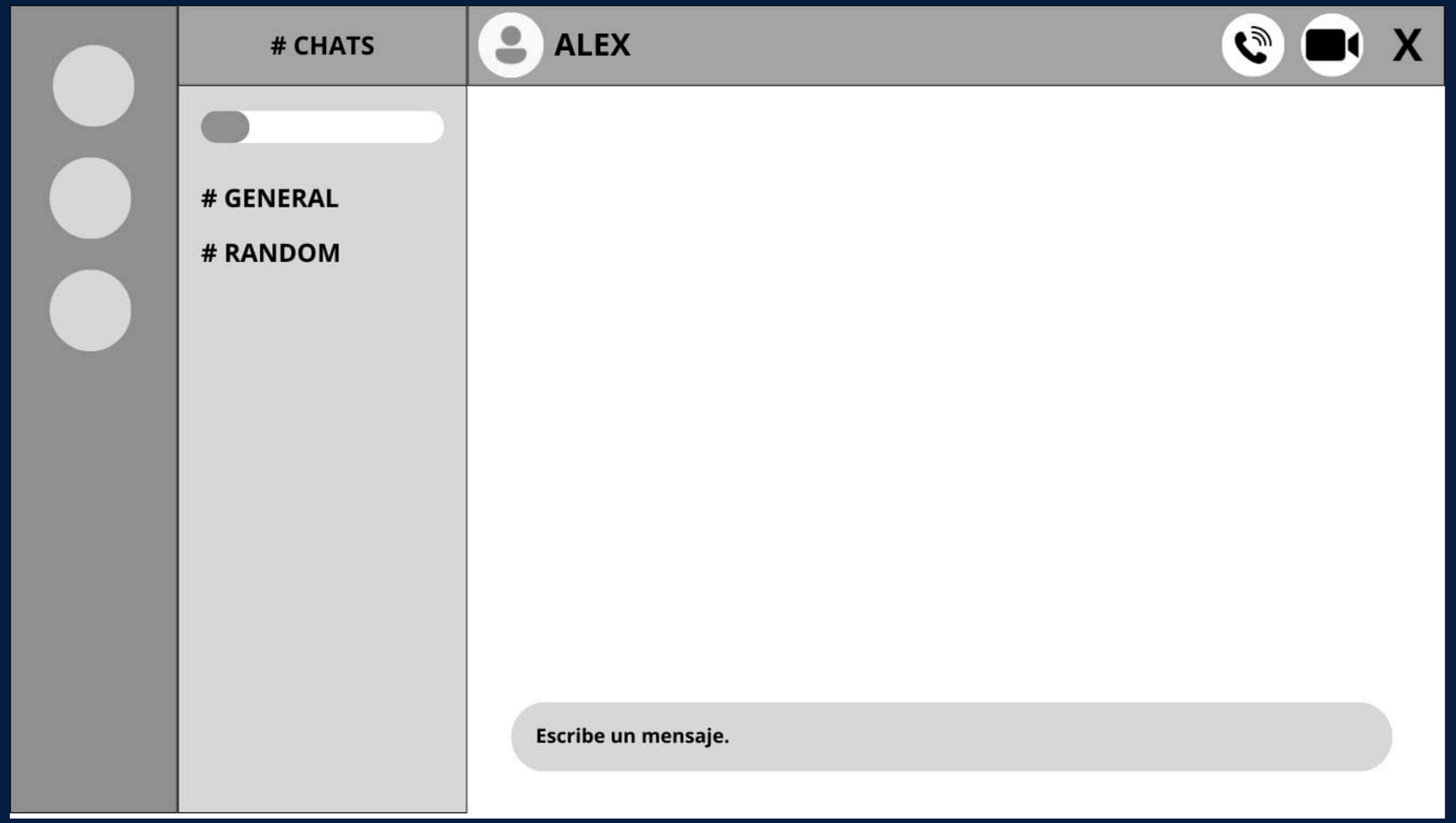
1. Users can choose between light and dark mode themes.
2. Users can customize chat backgrounds and font size.
3. Changes are saved and applied automatically to the interface.

MOCK-UPS

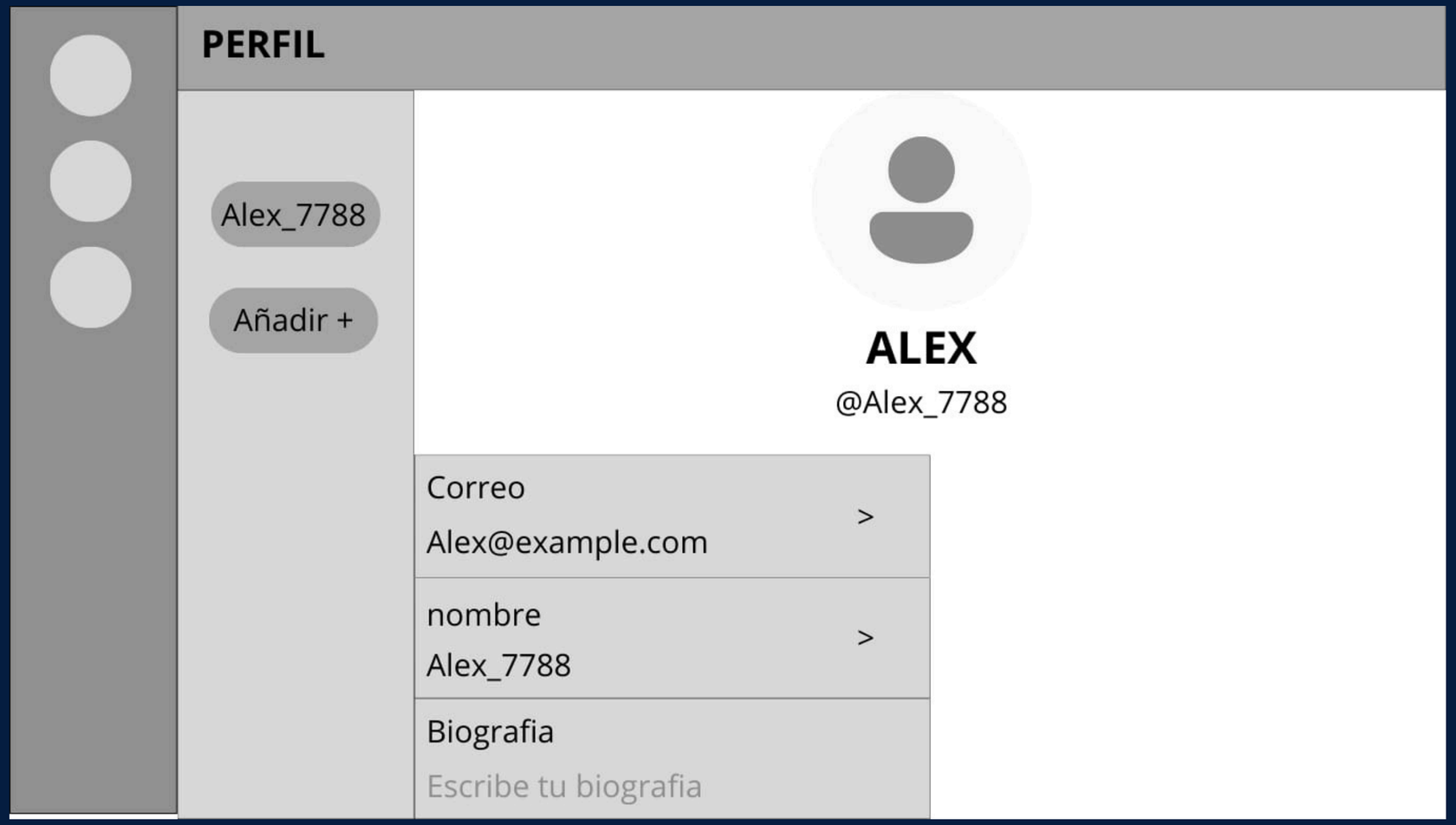
Sign-In


¡Bienvenido!
CORREO:
CONTRASEÑA:
INICIAR SESIÓN
Aún no tienes cuenta?
Registrate

Principal Page



Profile


PERFIL
Alex_7788
Añadir +
ALEX
@Alex_7788
Correo
Alex@example.com
>
nombre
Alex_7788
>
Biografia
Escribe tu biografia

CRC CARDS

Class: User

RESPONSIBILITIES	COLLABORATORS
• Register and log in • Edit profile (name, status, profile picture) • Add and remove contacts • Send and receive messages • Start and receive calls	• Contact List • Message • Call

Class: Contact

RESPONSIBILITIES	COLLABORATORS
• Represent another user in the contact list • Display contact name and status	• User

Class: Chat

RESPONSIBILITIES	COLLABORATORS
• Manage a conversation between two or more users • Store a list of messages • Display messages in order • Allow sending new messages within the chat	• User • Message

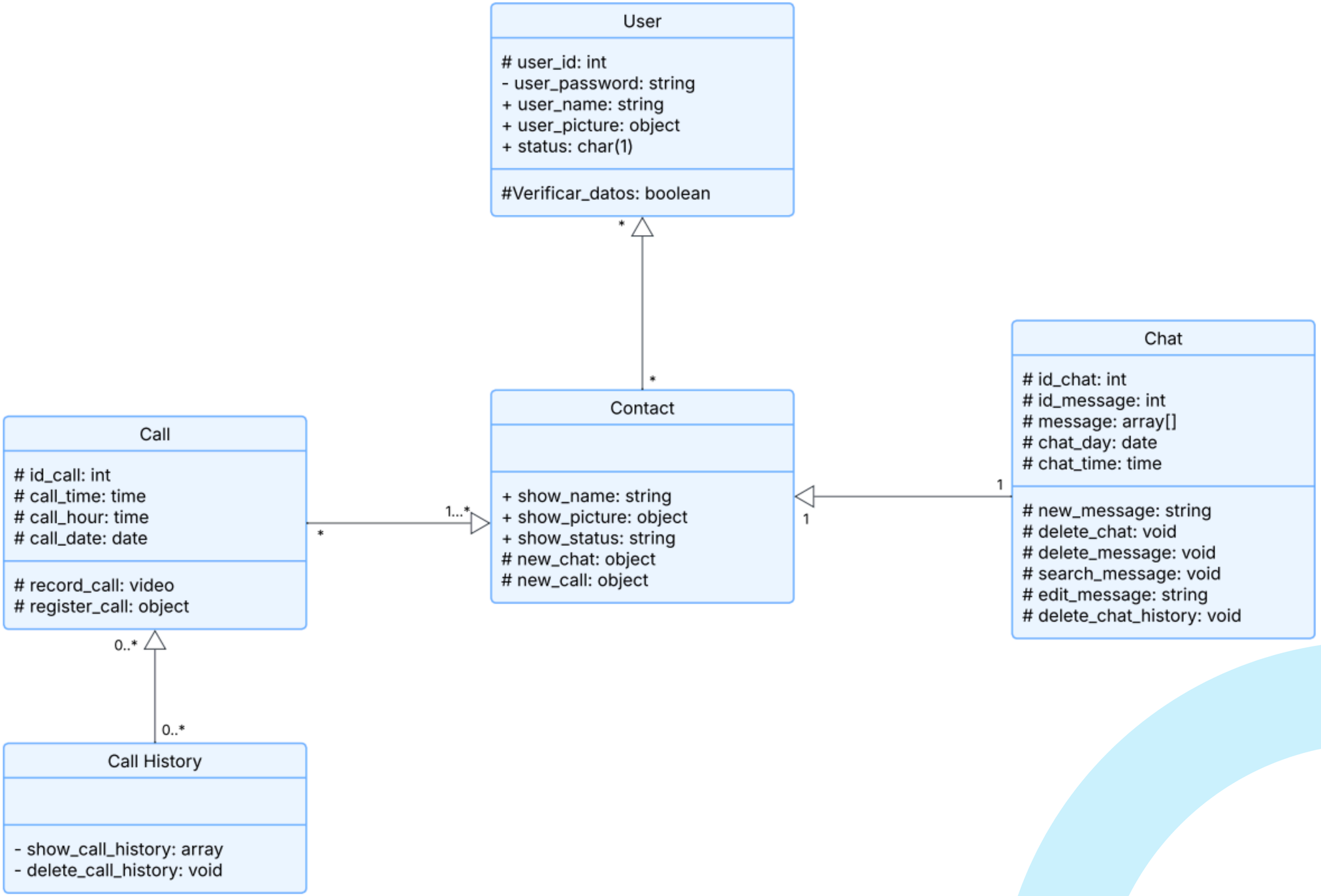
Class: Call

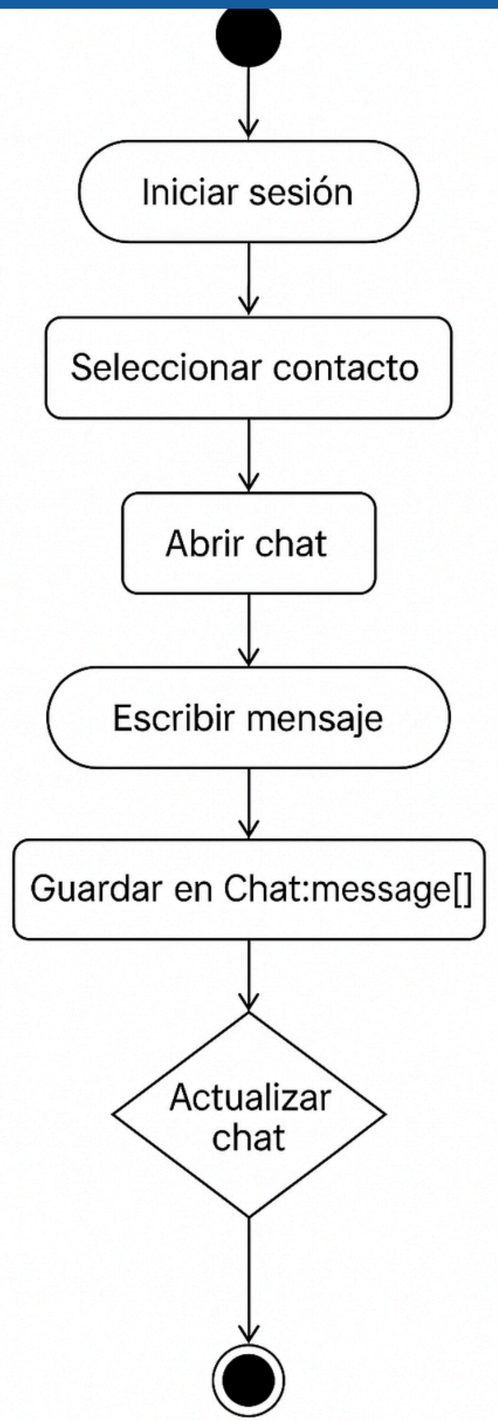
RESPONSIBILITIES	COLLABORATORS
• Start a call between two or more users • Track duration and participants • End the call	• User • Call History

Class: Call History

RESPONSIBILITIES	COLLABORATORS
• Register call information	• User • Message • Call

UML
Diagram


User
user_id: int
- user_password: string
+ user_name: string
+ user_picture: object
+ status: char(1)
#Verificar_datos: boolean
*
*
Contact
+ show_name: string
+ show_picture: object
+ show_status: string
# new_chat: object
# new_call: object
Chat
id_chat: int
id_message: int
message: array[]
chat_day: date
chat_time: time
new_message: string
delete_chat: void
delete_message: void
search_message: void
edit_message: string
delete_chat_history: void
1
1
Call
id_call: int
call_time: time
call_hour: time
call_date: date
record_call: video
register_call: object
1...*
*
0..*
0..*
Call History
- show_call_history: array
- delete_call_history: void



HOW DO WE APPLY OOP CONCEPTS

Encapsulation

The class Contact inherits from User, meaning it automatically gets all the attributes and methods from User (like login and logout). This avoids repeating code and allows Contact to add its own specific features, like status.

Polymorphism

Polymorphism lets different classes use the same method name but with different behavior. In this case, allows Contact to override methods from User using the same method names (like login) but with different behavior. This makes it possible to treat different objects similarly while letting them respond in their own way.

INITIAL CODE FRAGMENTS

```python
class user:
    def __init__(self, username, password, email, age):
        self.username = username
        self.password = password
        self.email = email
        self.age = age

    def login(self):
        print("Iniciando sesión...")
    def logout(self):
        print("Cerrando sesión...")

username = input("Ingrese su nombre de usuario: ")
password = input("Ingrese su contraseña: ")
email = input("Ingrese su email: ")
age = input("Ingrese su edad: ")

user1 = user(username, password, email, age)
```

```python
print(f"""
        Datos del usuario: \n\n
        Nombre de usuario: {user1.username} \n
        Contraseña: {user1.password} \n
        Email: {user1.email} \n
        Edad: {user1.age} \n
        """)


while True:
    login = input()
    if (login.lower() == "login"):
        user1.login()


    while True:
        logout = input()
        if (logout.lower() == "logout"):
            user1.logout()
```

INITIAL CODE FRAGMENTS

```python
class User:
    def _init_(self, username, password, email, age):
        self.username = username
        self.password = password
        self.email = email
        self.age = age

    def login(self):
        print(f"{self.username} ha iniciado sesión.")

    def logout(self):
        print(f"{self.username} ha cerrado sesión.")
class Contact(User):
    def _init_(self, username, password, email, age, status):
        super()._init_(username, password, email, age)
        self.status = status

    def show_info(self):
        print(f"Contacto: {self.username} - Estado: {self.status}")
```

INITIAL CODE FRAGMENTS

```
class User:
    def _init_(self, username, password, email, age):
        self.username = username
        self.password = password
        self.email = email
        self.age = age

    def login(self):
        print(f"{self.username} ha iniciado sesión.")

    def logout(self):
        print(f"{self.username} ha cerrado sesión.")
class Call:
    def _init_(self, participants):
        self.participants = participants
        self.active = False

    def start(self):
```