



APPCHAT

Fecha 11 de mayo de 2025

URL del repositorio del github:

<https://github.com/alexcarrionn/TDS.git>

Hecho por:

Juan Pedro Jiménez Dato

juanpedro.jimenezd@um.es

Alejandro Carrión Jordán

a.carrionjordan@um.es

Tecnologías del
desarrollo del
software

INDICE

Contenido

INDICE.....	1
1. Introducción	2
2. Diagrama de Usuario e historias de usuario	3
2.1 Diagrama de clases del dominio	3
2.2 Historias de usuario	4
3. Diagrama de secuencia para la operación añadir un nuevo contacto a un grupo	8
4. Arquitectura de la aplicación	9
5. Patrones de diseño	11
5.1 Patrón strategy	11
5.2 Patrón Método Factoría	12
5.3 Patrón Adaptador	12
6. Manual de Usuario	14
7. Observaciones finales	26



1. Introducción

El documento contiene la memoria de proyecto de la asignatura Tecnologías de Desarrollo del Software (TDS), tiene como objetivo detallar el diseño y la implementación de los componentes de la aplicación **APPCHAT**.

En este documento podremos encontrar información acerca de los siguientes componentes relacionados con el trabajo:

- Historias de usuario
- Diagrama utilizado para el proyecto
- Arquitectura de la aplicación y algunas medidas tomadas
- Patrones utilizados en el desarrollo del proyecto

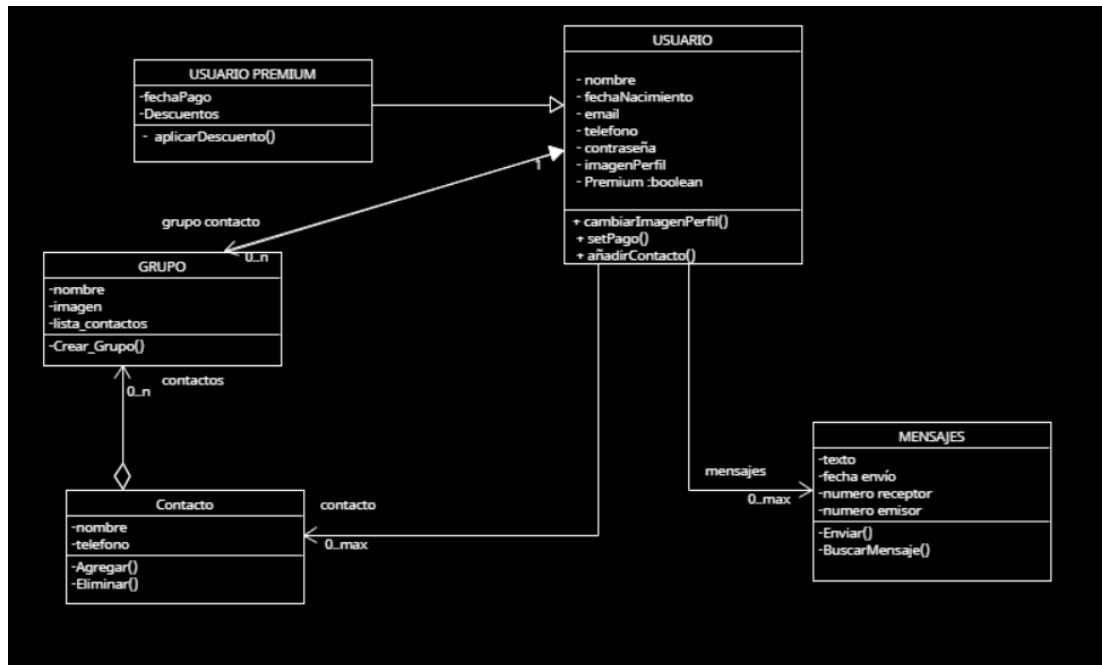
Este documento también incluirá un manual de uso con el que el usuario podrá saber mejor cómo funciona la aplicación desarrollada.



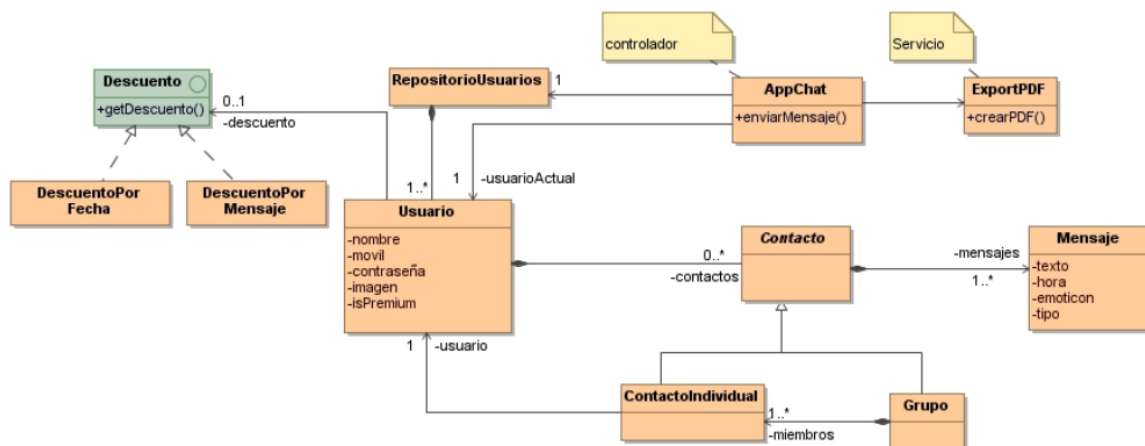
2. Diagrama de Usuario e historias de usuario

2.1 Diagrama de clases del dominio

El diagrama de clases del dominio que utilizamos para poder desarrollar el proyecto de AppChat en un principio fue el siguiente:



Sin embargo, tras la primera entrega, un gran debate entre nosotros y tras los consejos del profesor decidimos utilizar el siguiente diagrama (aportado por los profesores):



2.2 Historias de usuario

Historia de Usuario 1: Registro de un nuevo usuario

Como usuario no registrado,

quiero poder registrarme en el sistema proporcionando mis datos personales,

para poder acceder a la aplicación y utilizar sus funcionalidades.

Criterios de Verificación:

- El formulario de registro debe solicitar al menos los siguientes datos: nombre, apellidos, fecha de nacimiento (opcional), número de teléfono, imagen de perfil (opcional), estado y contraseña.
- El usuario debe repetir la contraseña para verificarla.
- El sistema debe validar que todos los campos obligatorios estén completos.
- Si el registro es exitoso, el usuario debe recibir una confirmación y ser redirigido a la página de inicio de sesión.
- Si el número de teléfono ya está registrado, el sistema debe mostrar un mensaje de error adecuado.

2. Historia de Usuario: Inicio de sesión

Como usuario registrado,

quiero iniciar sesión en la aplicación utilizando mi número de teléfono y contraseña,

para acceder a mis contactos y mensajes.

Criterios de Verificación:

- El formulario de inicio de sesión debe permitir introducir el número de teléfono y la contraseña.
- El sistema debe validar que las credenciales coincidan con las de un usuario registrado.



- Si el inicio de sesión es exitoso, el usuario debe ser redirigido a la pantalla principal de la aplicación.
- Si el número de teléfono o la contraseña no coinciden, el sistema debe mostrar un mensaje de error.
- Si el campo de número de teléfono o el campo de contraseña están vacíos, el sistema debe mostrar un mensaje de error.

3. Historia de Usuario: Añadir contacto

Como usuario registrado,

quiero añadir un nuevo contacto a mi lista de contactos,

para poder enviarle mensajes fácilmente.

Criterios de Verificación:

- El usuario debe poder añadir un contacto proporcionando un número de teléfono y un nombre asociado.
- El sistema debe validar que el número de teléfono no esté ya en la lista de contactos del usuario.
- Si el número de teléfono no existe en el sistema, el sistema debe notificar al usuario con un mensaje de error.
- El nuevo contacto debe aparecer en la lista de contactos del usuario una vez añadido con éxito.

4. Historia de Usuario: Crear grupo

Como usuario,

quiero crear un grupo de contactos,

para facilitar el envío de mensajes a varios contactos al mismo tiempo.



Criterios de Verificación:

- El sistema debe permitir al usuario crear un grupo proporcionando un nombre para el grupo y opcionalmente una imagen, si no se le asigna una imagen, el sistema le proporcionará una imagen por defecto.
- El usuario debe poder añadir varios contactos existentes al grupo.
- El sistema debe validar que el nombre del grupo no esté vacío.
- El grupo recién creado debe aparecer en la lista de contactos del usuario.

5. Historia de Usuario: Enviar y recibir mensajes

Como usuario registrado,

quiero enviar y recibir mensajes a/de mis contactos,

para poder comunicarme con ellos de forma rápida y eficiente.

Criterios de Verificación:

- El usuario debe poder seleccionar un contacto o un grupo de su lista de contactos para enviar un mensaje.
- El mensaje debe incluir texto o un emoticono, y se debe registrar con la fecha y hora de envío.
- El mensaje debe aparecer inmediatamente en la ventana de conversación una vez enviado.
- Si se recibe un mensaje de un usuario que no es un contacto, se debe mostrar el número y el estado del contacto que mandó el mensaje.
- Si el emisor no está en la lista de contactos del usuario, el sistema debe permitir al usuario añadirlo como contacto.
- Los mensajes enviados a un grupo se envían individualmente a cada uno de los contactos del grupo



6. Historia de Usuario: Convertirse en usuario premium

Como usuario registrado,

quiero poder convertirme en usuario premium pagando una suscripción,

para acceder a funciones adicionales como la exportación de mensajes.

Criterios de Verificación:

- El sistema debe permitir al usuario registrarse como premium mediante el pago de una suscripción anual.
- El sistema debe aplicar descuentos automáticos basados en la fecha de registro o el número de mensajes enviados.
- El usuario premium debe tener acceso a funciones adicionales como la exportación de mensajes en PDF.
- El usuario debe poder cancelar la suscripción premium en cualquier momento, y el sistema debe revertir el acceso a las funcionalidades premium tras la cancelación.

7. Historia de Usuario: Buscar mensajes

Como usuario registrado,

quiero buscar mensajes por fragmento de texto, por fecha de envío o por tipo del mensaje,

para encontrar fácilmente los mensajes que necesito.

Criterios de Verificación:

- El sistema debe permitir buscar mensajes enviados o recibidos por el usuario filtrando por fragmento de texto, por fecha de envío o por tipo del mensaje.
- El sistema debe permitir combinar varios criterios de búsqueda (por ejemplo, texto y tipo).



8. Historia de Usuario: Exportar mensajes a PDF (solo premium)

Como usuario premium,

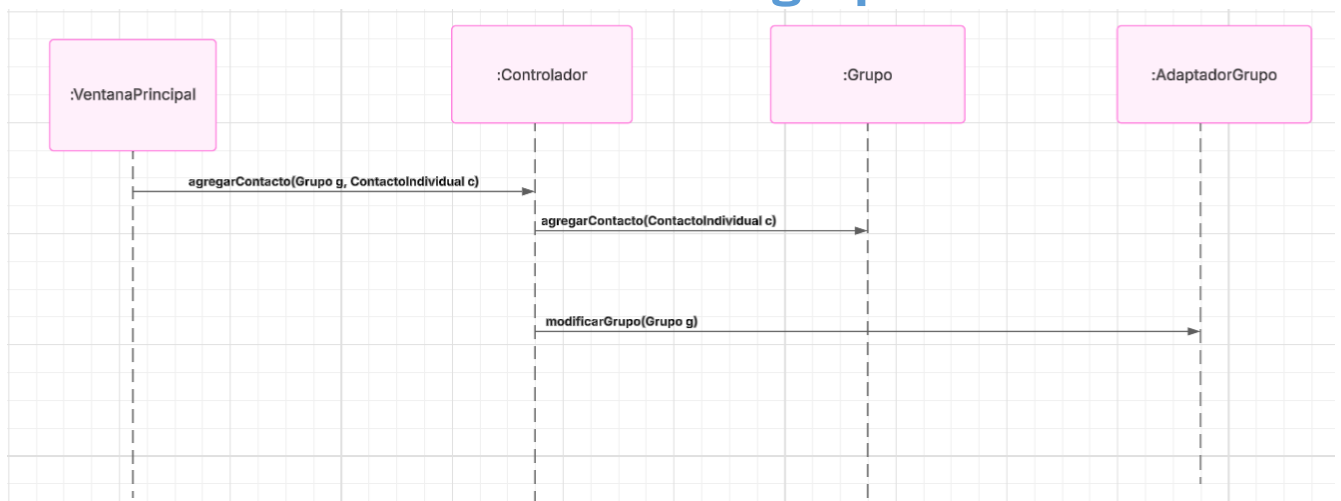
quiero exportar mis conversaciones con todos los contactos y grupos a un archivo PDF,

para tener un registro de los mensajes intercambiados.

Criterios de Verificación:

- El archivo PDF debe incluir los nombres de los participantes, número de teléfono de estos, el contenido del mensaje, y la fecha y hora de cada mensaje.
- El archivo PDF debe generarse correctamente y descargarse en el dispositivo del usuario

3. Diagrama de secuencia para la operación añadir un nuevo contacto a un grupo



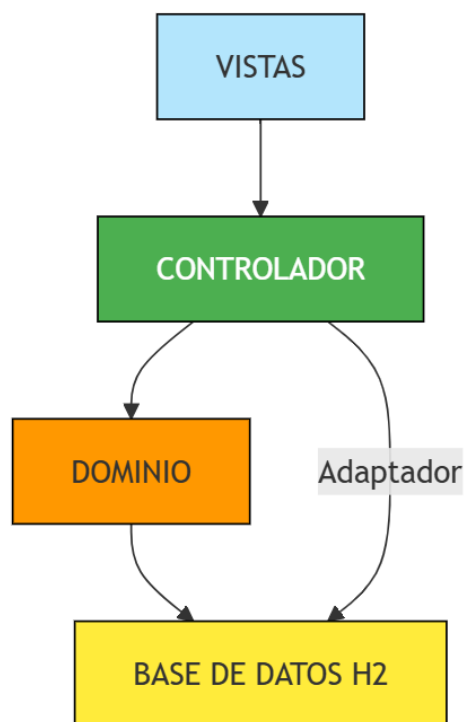
Este diagrama muestra cómo se añade un Contacto a un grupo dentro de la aplicación. El proceso inicia en la ventana principal (interfaz gráfica) donde se llama a la función `agregarContacto` en la función `anadirAlGrupo()`. Esta función contactará con el controlador, quien delegará al grupo para que agregue el contacto a la lista de contactos del grupo. Finalmente, el controlador notificará a `adaptadorGrupo` para que este actualice la base de datos.



4. Arquitectura de la aplicación

Nuestra aplicación consta de 24 clases de entre las que se conforman por las ventanas, dominio, adaptadores para las diferentes entidades que se guardan en la persistencia, los componentes para cargar los mensajes y los contactos y el controlador que es el que hará de intermediario entre las vistas y el dominio.

Una imagen conceptual donde podremos ver cómo es la arquitectura de la aplicación es la siguiente:



Esta arquitectura la diseñamos con el objetivo de poder cumplir los principios del diseño establecido por los patrones GRASP dados en las clases de teoría. Pues así podemos garantizar las correctas responsabilidades que van a tener cada uno de los componentes de nuestro sistema.

Como se puede ver en la imagen anterior el controlador actúa como intermediario entre las vistas y el dominio, coordinando así las acciones y evitando que las vistas manejen la lógica de negocio de nuestra aplicación. Las vistas solo se encargarían de la interacción de la aplicación con el usuario, es decir, mostrar y capturar datos, mientras que las operaciones



más complejas, es decir, aquellas que requieran un acceso a la base de datos o al sistema de persistencia, se delegarían a las clases del modelo o a los adaptadores DAO.

Y por último, dominio o modelo, el cual es el que contiene la lógica de negocio y los datos sin depender de la vista o del controlador. Los adaptadores nos ayudarían a abstraer el acceso a la base de datos H2, ocultando esos detalles técnicos y permitiendo cambiar el sistema de persistencia sin tener la necesidad de cambiar la vista o el modelo.

Una de las decisiones que tuvimos que hacer fue la implementación de “ContactoListCellRenderer”, la cual, el funcionamiento principal que tiene es el renderizado personalizado de las celdas de los usuarios, ya que define cómo se muestra cada uno de los contactos, ya sean **ContactoIndividual** o **Grupo** en la lista de la ventana principal, además de combinar imágenes y texto en un diseño específico y la estructura visual.

Como hicimos con los contactos también tuvimos que hacerlo con los mensajes, aquí decidimos crear la clase “MensajeCellRender” que como ya hemos explicado para la de contactos, la funcionalidad principal que tiene es el renderizado personalizado de mensajes y el diseño visual de estos.

Hay que comentar que tuvimos que crear una clase interna llamada ChatBurbujas, que implementa JPanel, esta clase de lo que se encarga es de configurar el panel que va a contener las burbujas con los mensajes de manera personalizada y luego tiene un método para poder añadir una burbuja de mensaje.

Como resumen se podría decir que la implementación de la aplicación se basa en una clara separación de responsabilidades. Además, se podría decir que, gracias a la modularidad de este proyecto, se garantiza que el sistema sea fácil de mantener y de poder ampliar en un futuro.



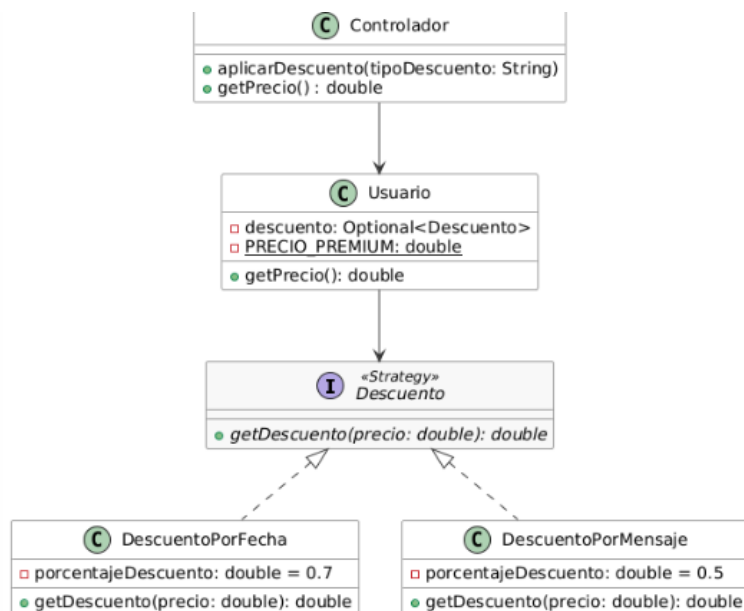
5. Patrones de diseño

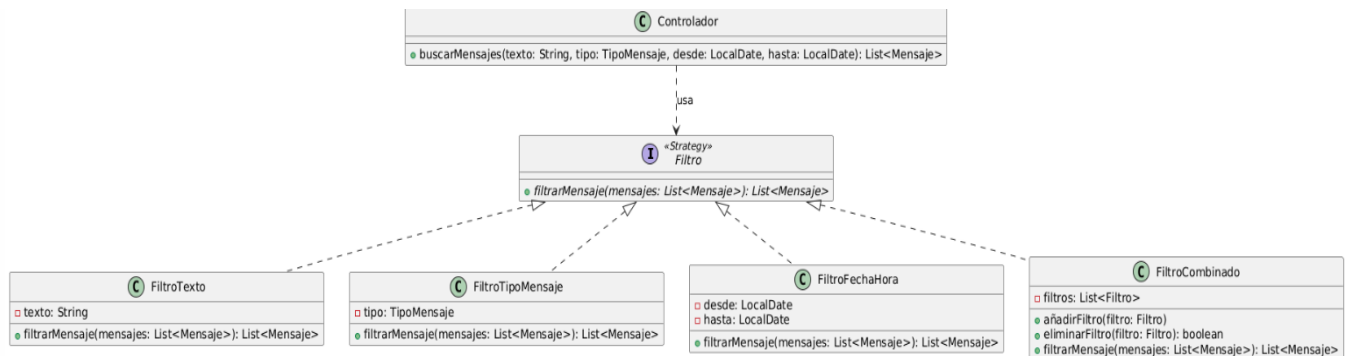
5.1 Patrón strategy

El controlador usa el patrón estrategia para gestionar tanto el descuento que se le aplicará a los usuarios, como el tipo de filtro a utilizar para poder filtrar los mensajes. El controlador se basará en unas interfaces llamadas **descuentoStrategy** y **filtrosStrategy**, donde se encuentran Filtro y Descuento. Estos definen los métodos **getDescuento(double precio)** y **filtrarMensaje(List<Mensaje> mensajes)**.

En cuanto al descuento podemos decir que las diferentes estrategias, como **DescuentoPorFecha** y **DescuentoPorMensajes**, implementan esta interfaz, encapsulando la lógica específica de cada descuento en clases independientes.

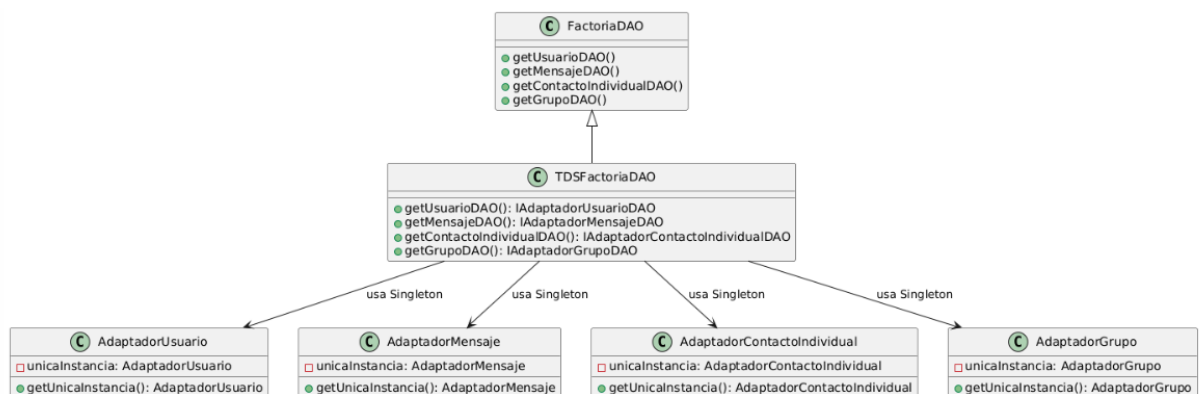
Además, en cuanto a los filtros podemos encontrar diferentes estrategias, como **FiltroTexto**, **FiltroFechaHora**, **FiltroCombinado**, **FiltroTipoMensaje**, implementan esta interfaz, encapsulando la lógica específica de cada descuento en clases independientes.





5.2 Patrón Método Factoría

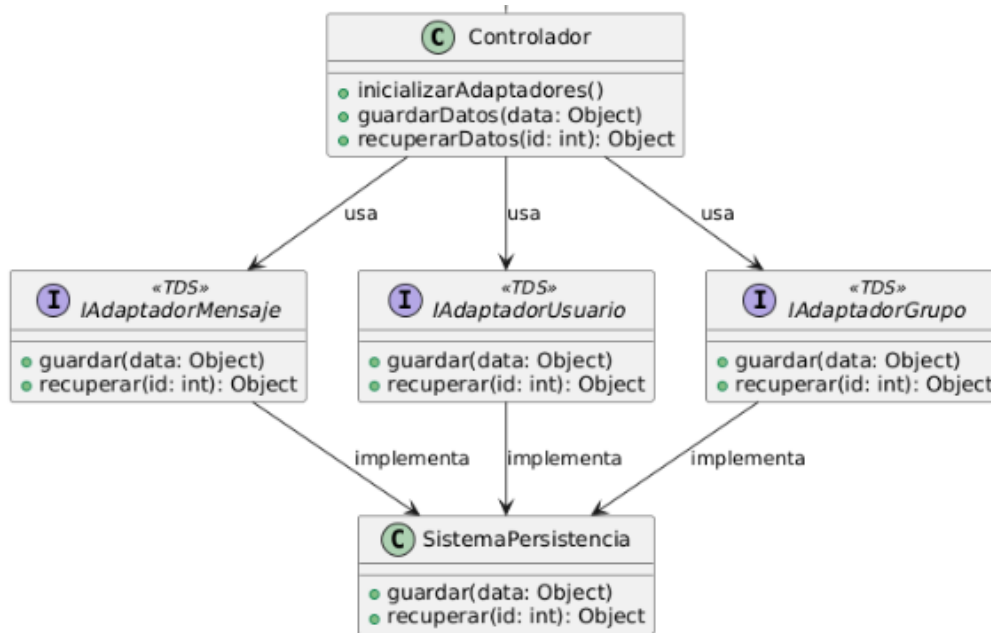
El patrón Método Factoría se usa en la clase **TDSFactoriaDAO** para poder crear y devolver instancias concretas de adaptadores DAO, como los que gestionan mensajes, usuarios, contactos o incluso grupos. Estos adaptadores, que siguen el patrón Singleton para poder garantizar que solo haya una única instancia de estos, permiten que el controlador obtenga fácilmente las clases necesarias sin preocuparse por cómo se crean. Este enfoque hace que el sistema sea mucho más flexible y mucho más fácil de mantener.



5.3 Patrón Adaptador

El método de **inicializarAdaptadores** utiliza el patrón Adaptador al emplear los adaptadores DAO como intermediarios entre el controlador y el sistema de persistencia. Cada adaptador traduce las operaciones del controlador al formato requerido por el sistema de persistencia, y viceversa, asegurando que el controlador no dependa directamente de los detalles como almacenar los datos, recuperarlos, etc... Esto hace que el sistema sea mucho más flexible y fácil de mantener, ya que permite cambiar el sistema de persistencia sin afectar la lógica del programa.





El código implementa un adaptador para usuarios, mensajes, grupos, contactos individuales, que gestionan estas entidades en un sistema de persistencia utilizando el patrón DAO. Además de todo eso se utiliza el patrón Singleton para poder asegurar que solo haya una instancia de los adaptadores, facilitando el acceso de forma eficiente a los datos. El patrón DAO es un diseño que abstrae y encapsula las operaciones de acceso a datos, lo que nos proporciona una interfaz común para poder realizar diferentes tareas que son referentes a la persistencia sin necesidad de que el sistema conozca los detalles de la implementación de esta. En este caso, el patrón DAO se utiliza en los diferentes adaptadores para poder gestionar la persistencia de mensajes, usuarios, contactos individuales y grupos, separando la lógica de acceso a la persistencia de la lógica del sistema. Además, se utiliza un pool de objetos para mejorar el rendimiento, evitando accesos repetidos a la persistencia. Esto hace que el sistema sea más modular, fácil de mantener y que sea mas eficiente hacer cambios en la capa de la persistencia

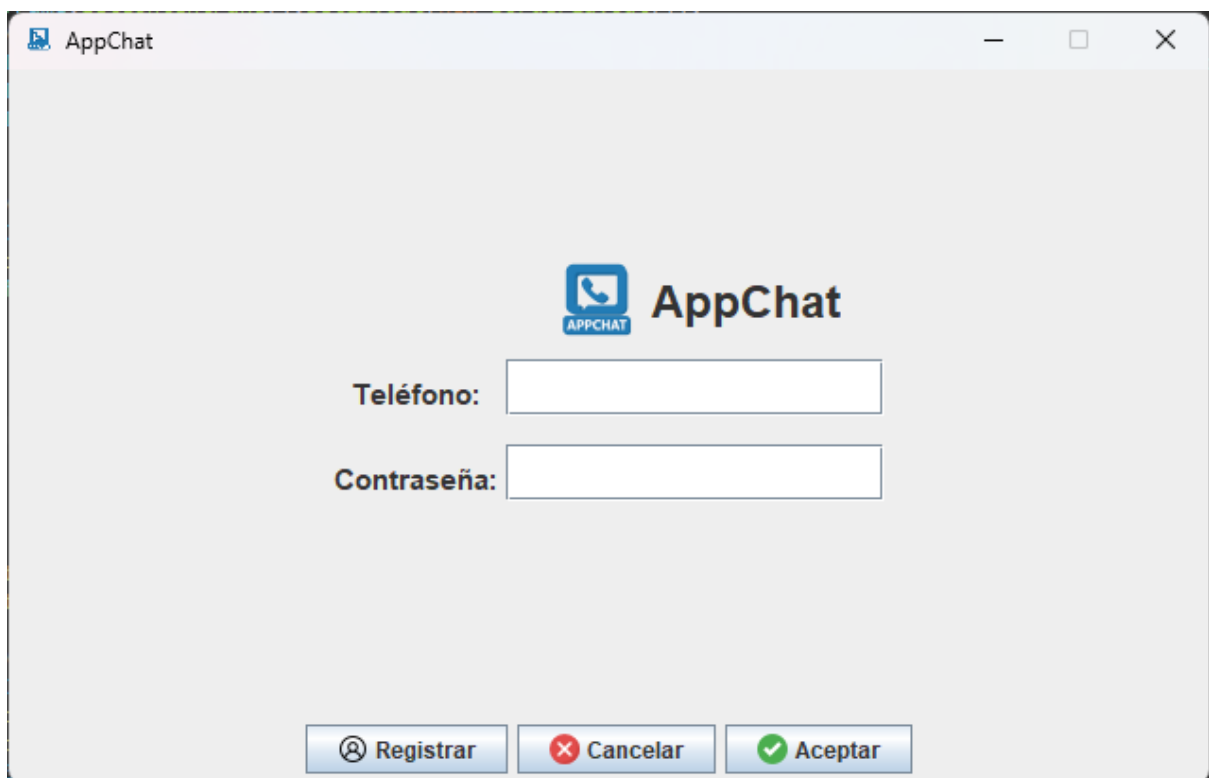


6. Manual de Usuario

Lo primero que debemos hacer para lanzar la aplicación es iniciar el servidor de persistencia para eso lanzamos una consola en la carpeta donde lo tengamos y ponemos el siguiente comando:

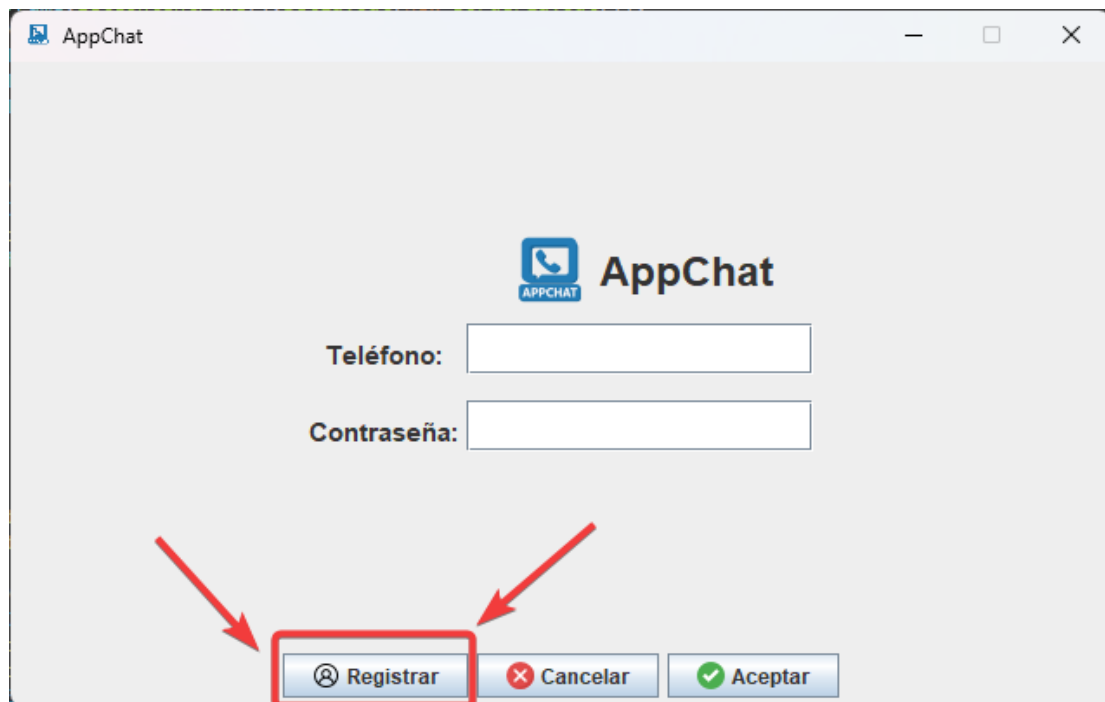
java -jar .\ServidorPersistenciaH2.jar

Una vez hecho esto ya podemos pasar a lo siguiente que es lanzar la aplicación, para ello tenemos que ejecutar la clase Lanzador:



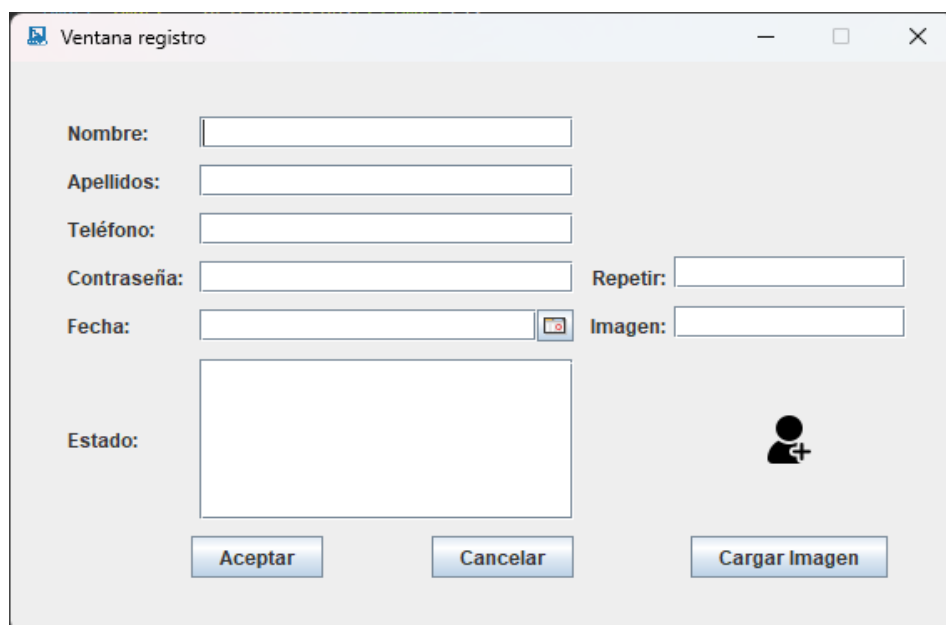
Esta es nuestra ventana de login, en ella podemos iniciar sesión o si aún no estamos registrados pulsar el botón para ello:





A screenshot of a Windows application window titled "AppChat". The window has a light gray background. At the top center, there is a blue square icon with a white telephone handset and the text "APPCHAT" below it, followed by the text "AppChat" in a bold, black, sans-serif font. Below this, there are two white text input fields. The first is labeled "Teléfono:" and the second is labeled "Contraseña:". At the bottom of the window, there are three buttons: "Registrar" (with a user icon), "Cancelar" (with a red 'X' icon), and "Aceptar" (with a green checkmark icon). The "Registrar" button is highlighted with a red rectangular border, and two red arrows point towards it from the left and top-left.

Con lo que se nos abrirá nuestra siguiente ventana que es la de registro:



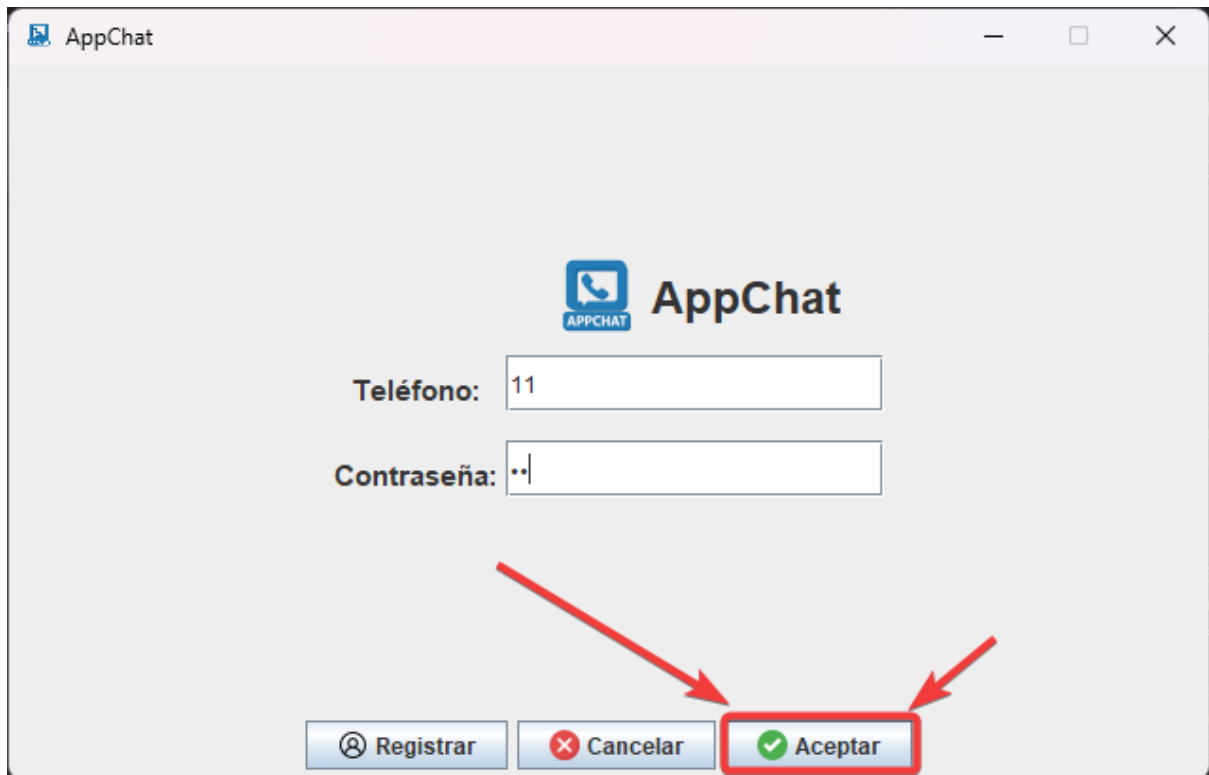
A screenshot of a Windows application window titled "Ventana registro". The window has a light gray background. It contains several white text input fields: "Nombre:", "Apellidos:", "Teléfono:", "Contraseña:", "Repetir:", "Fecha:", and "Imagen:". Below the "Fecha:" field is a small calendar icon. Below the "Imagen:" field is a small icon of a person with a plus sign. At the bottom of the window, there are three buttons: "Aceptar", "Cancelar", and "Cargar Imagen".

En esta ventana introducimos nuestros datos para registrarnos y de manera obligatoria para poder aceptar el registro satisfactoriamente deberemos introducir: Nombre, Apellidos,



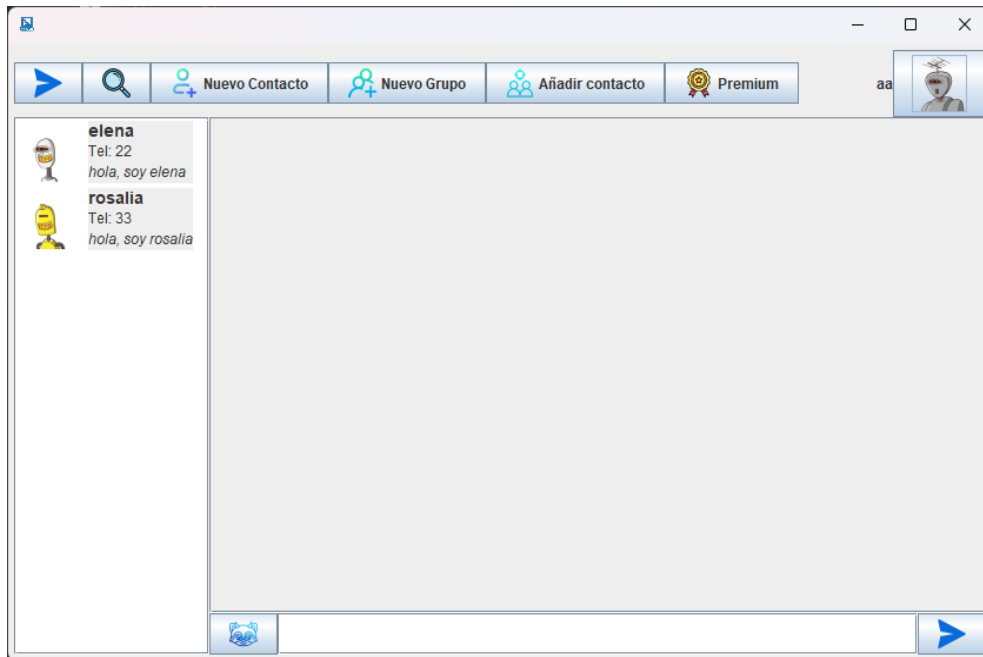
Teléfono, Contraseña dos veces y el estado. Por defecto la fecha será la actual y se nos asocia una imagen por defecto en caso de no introducir ninguna.

Lo siguiente que haremos será iniciar sesión:

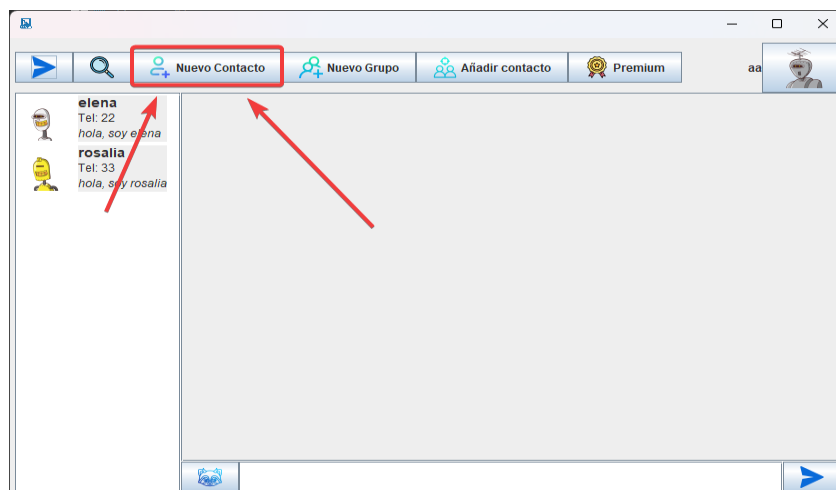


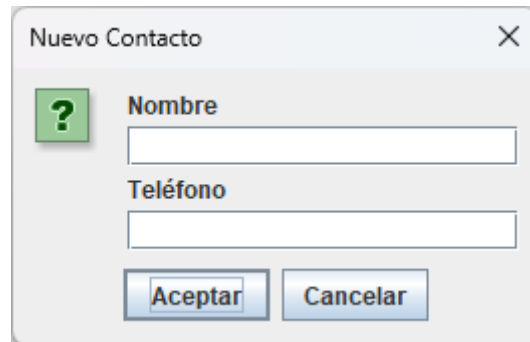
Una vez le demos a Aceptar, nos aparece la Ventana Principal de nuestra aplicación:





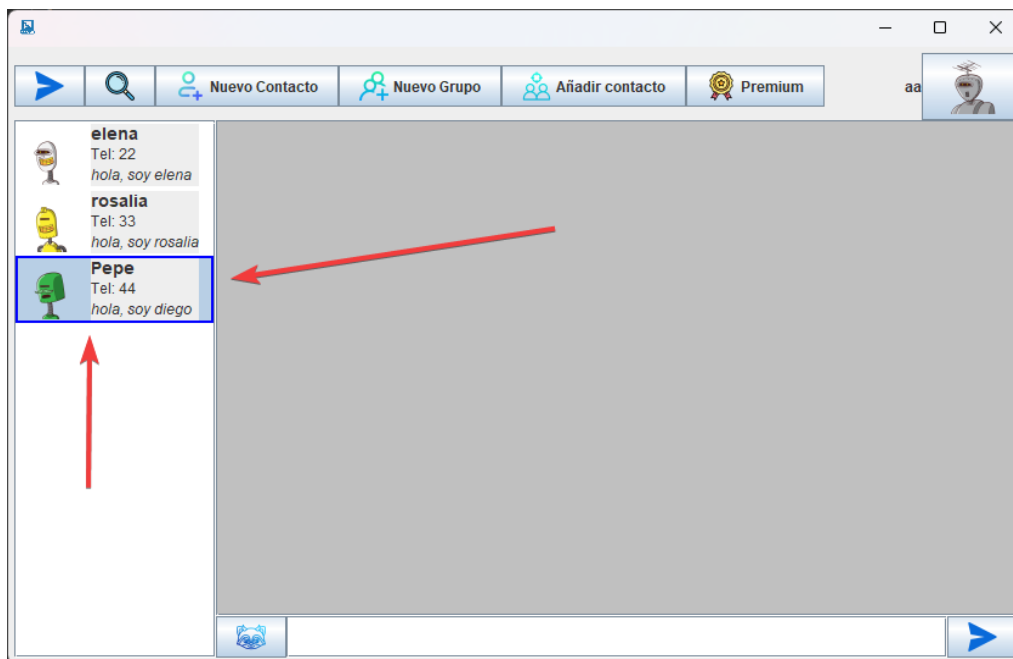
En este caso ya tenemos dos contactos agregados, pero para agregar un nuevo contacto que es lo primero que deberemos de hacer pulsamos el siguiente botón:





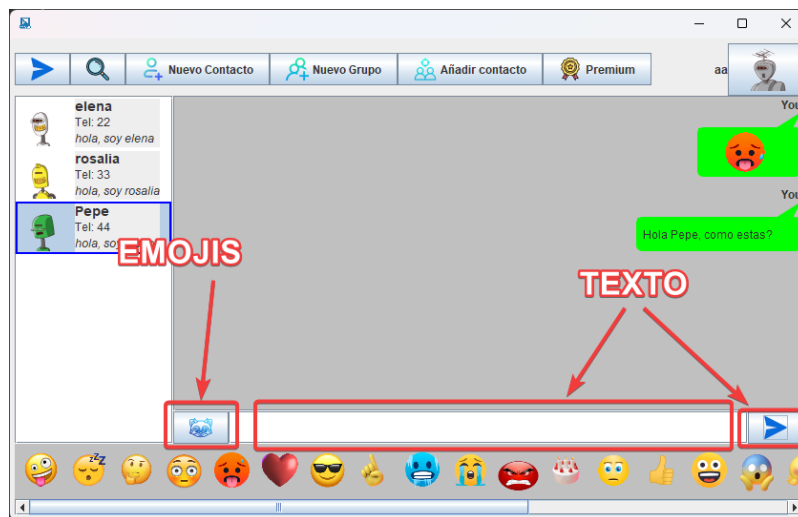
Se nos abrirá esta ventana en la que introduciremos el Nombre y el teléfono del contacto que queramos agregar. Por ejemplo, agregamos a Pepe con el número 44 y pulsamos aceptar.

Lo siguiente que podemos hacer es enviar un mensaje a Pepe, para ello clicamos en su tarjeta de contacto situada en el panel de contactos de la izquierda:

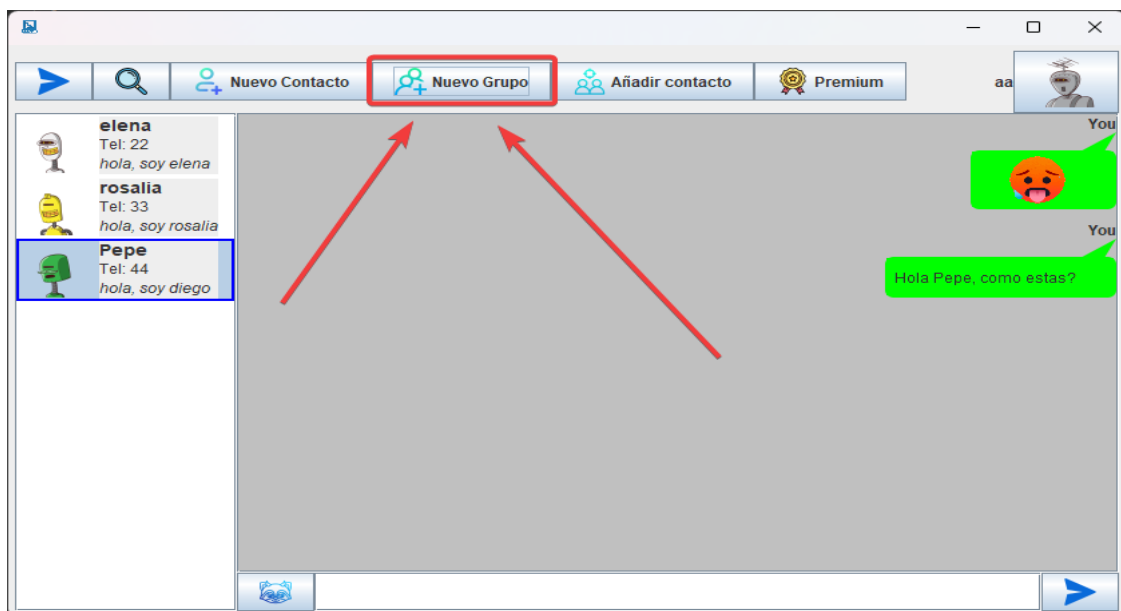


Una vez ahí ya podemos enviar mensajes de texto, si queremos mandar emoticonos pulsamos el botón de la izquierda y si queremos enviar mensajes escribimos en el panel y pulsamos el botón de la derecha:

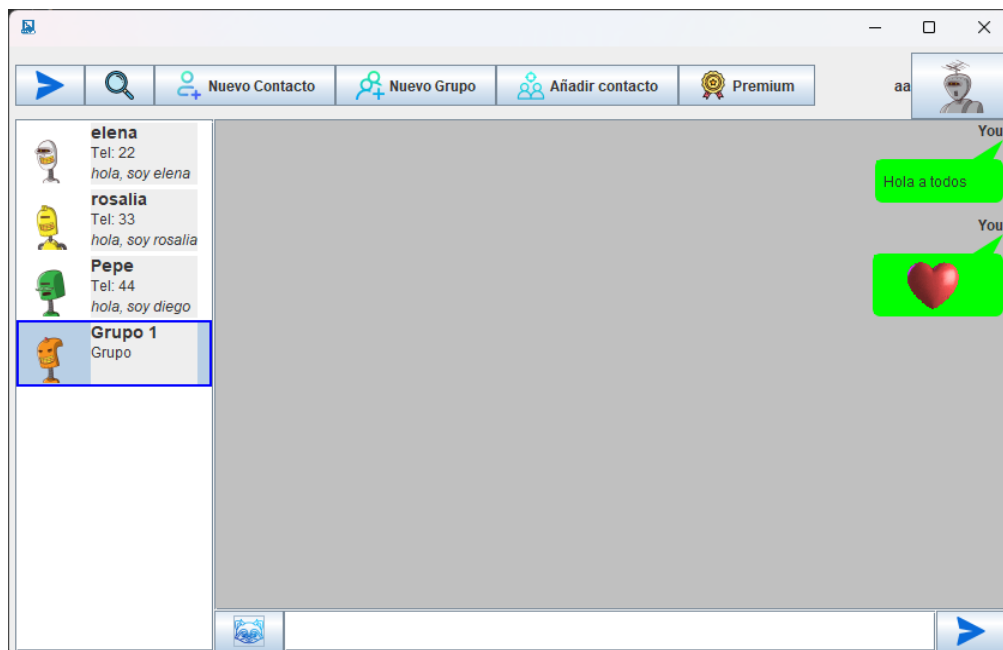




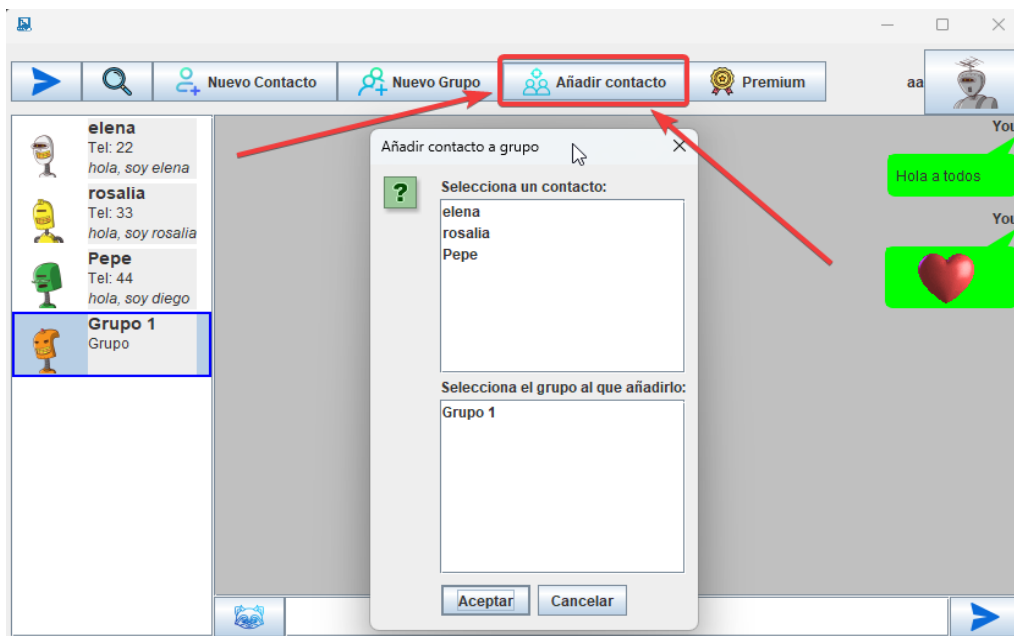
Tenemos otra opción disponible, la cual es crear grupos que funcionan como listas de difusión en la que agregamos contactos y cuando enviamos un mensaje les llega a todos el mismo mensaje, para ello pulsamos en el botón de Nuevo Grupo:



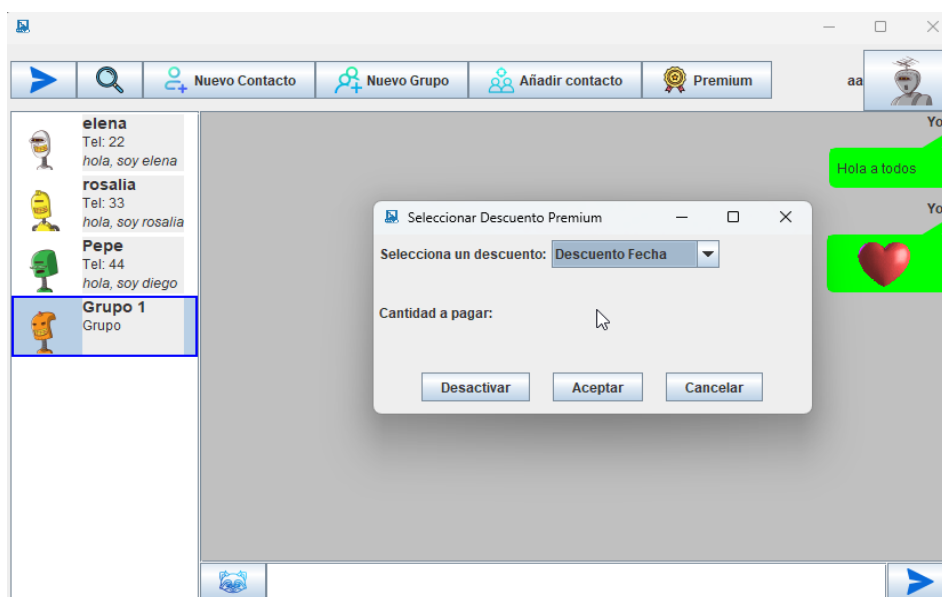
Seleccionamos los contactos que queramos, le ponemos un nombre al grupo, de manera opcional le podemos asignar una imagen a este grupo.



Y este mensaje les llega a todos los que hayamos agregado. En caso de que queramos agregar más contactos al grupo usaremos el botón a la derecha de Nuevo Grupo y podremos seleccionar contactos y al grupo al que queramos agregarlos:

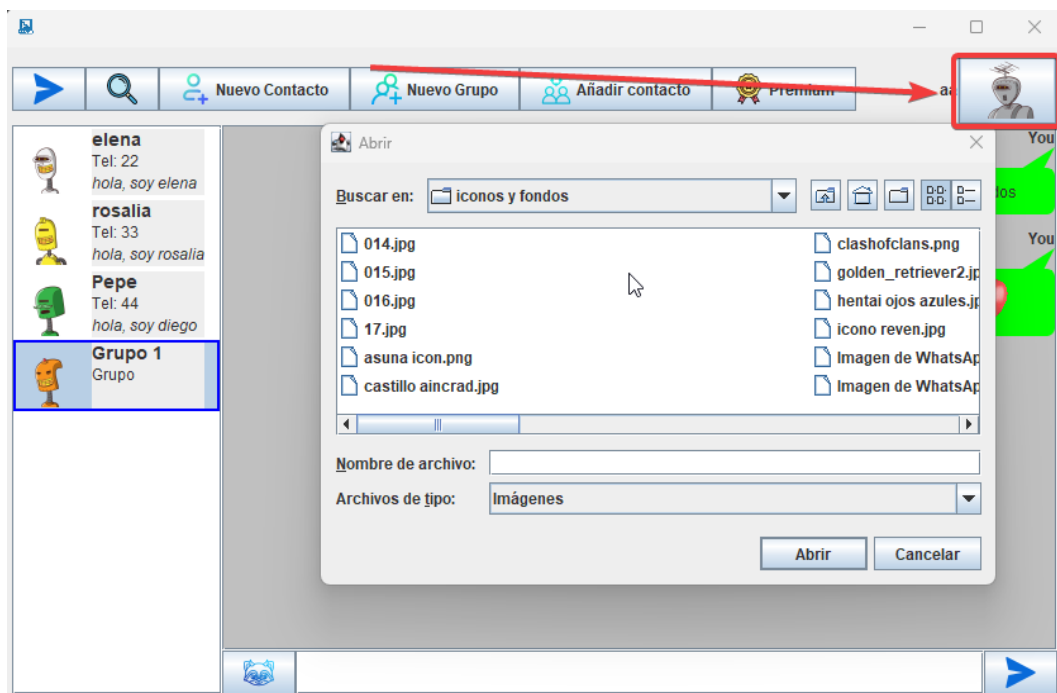


Por otro lado, tenemos la opción de convertirnos en usuarios Premium, para ello pulsaremos en el siguiente botón:

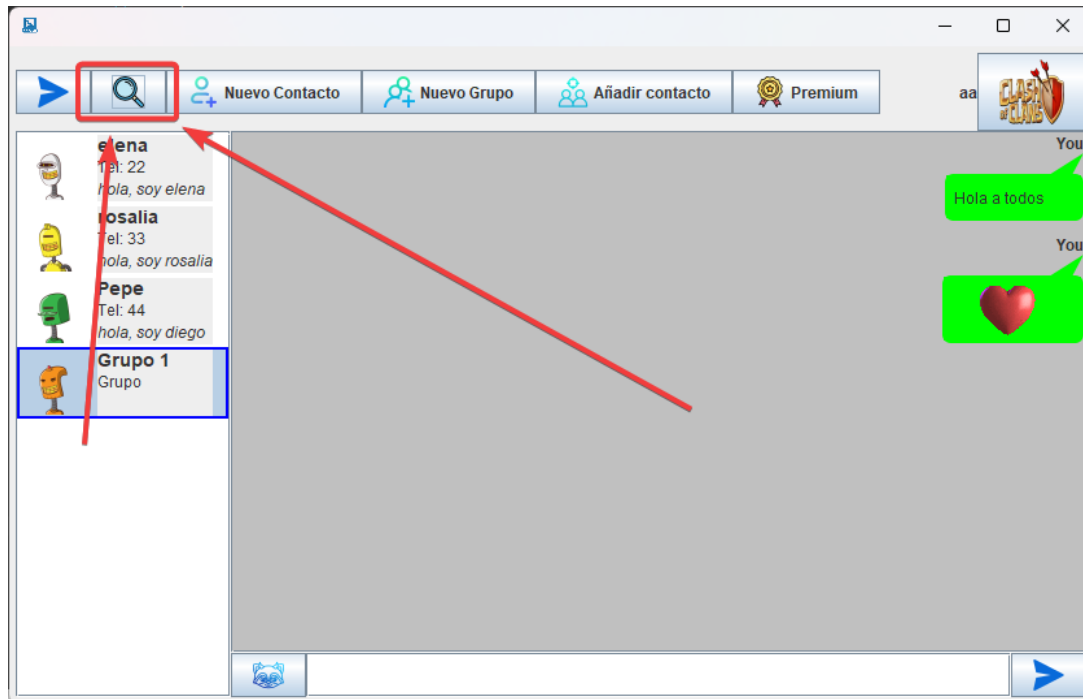


Podremos hacernos premium si tenemos algún tipo de descuento aplicable, ya sea el de fecha por habernos registrado en Navidad o el de Mensajes por haber enviado más de 100 mensajes.

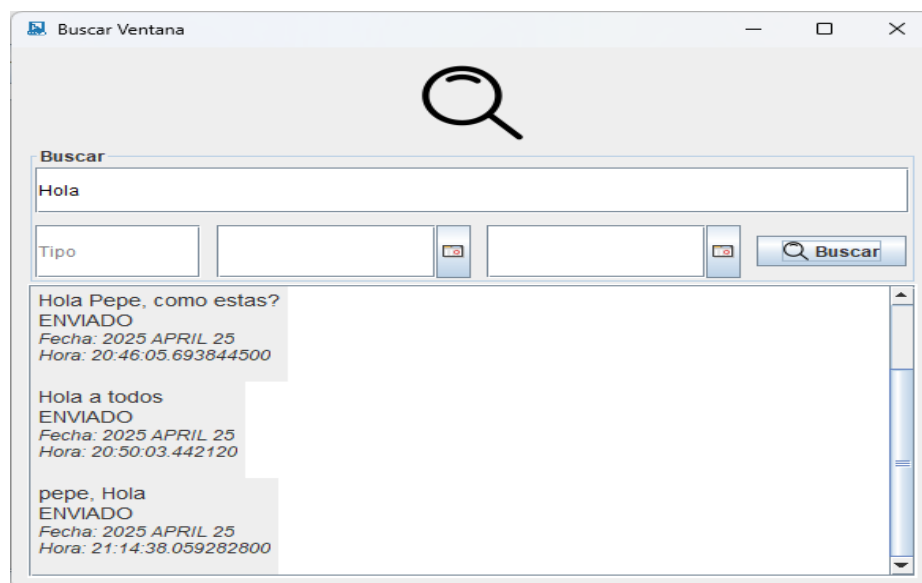
Por otro lado si clicamos en nuestra foto de perfil situado arriba a la derecha podremos cambiarla y se nos abrirá un explorador de archivos para seleccionar una imagen y ponerla:



Si nos vamos a los botones de la izquierda y le damos a la lupa tendremos un buscador de mensajes:

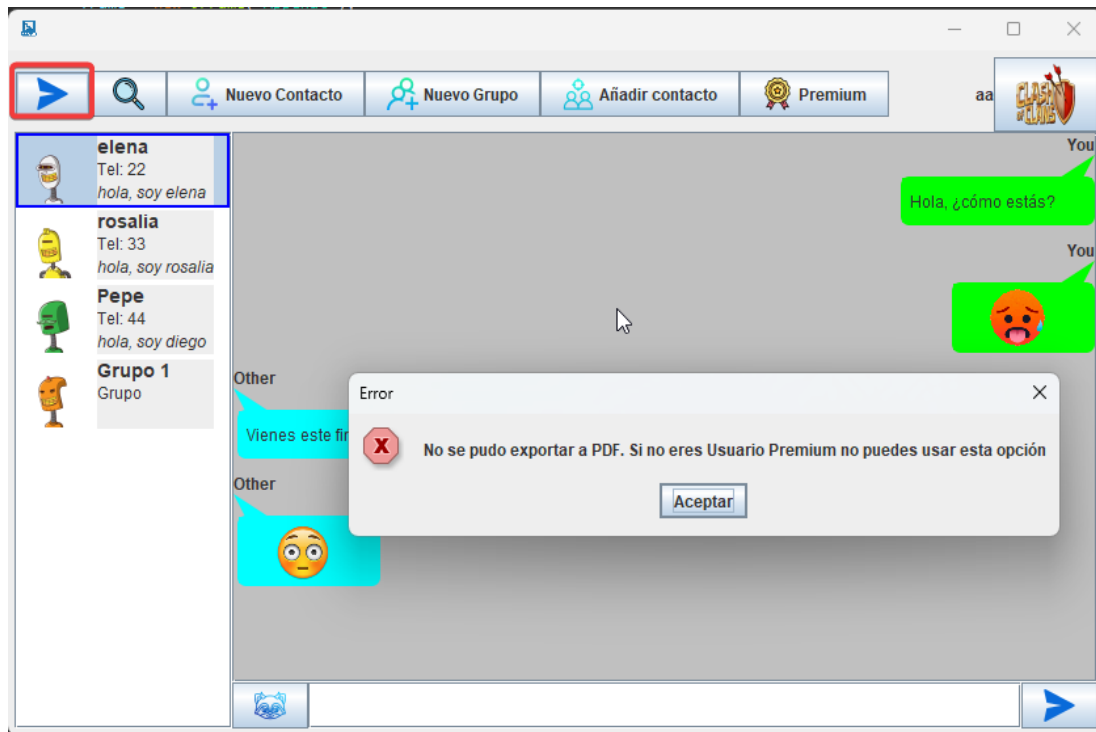


En el podremos buscar tanto por texto como por tipo de mensaje (ENVIADO o RECIBIDO) como por fecha:

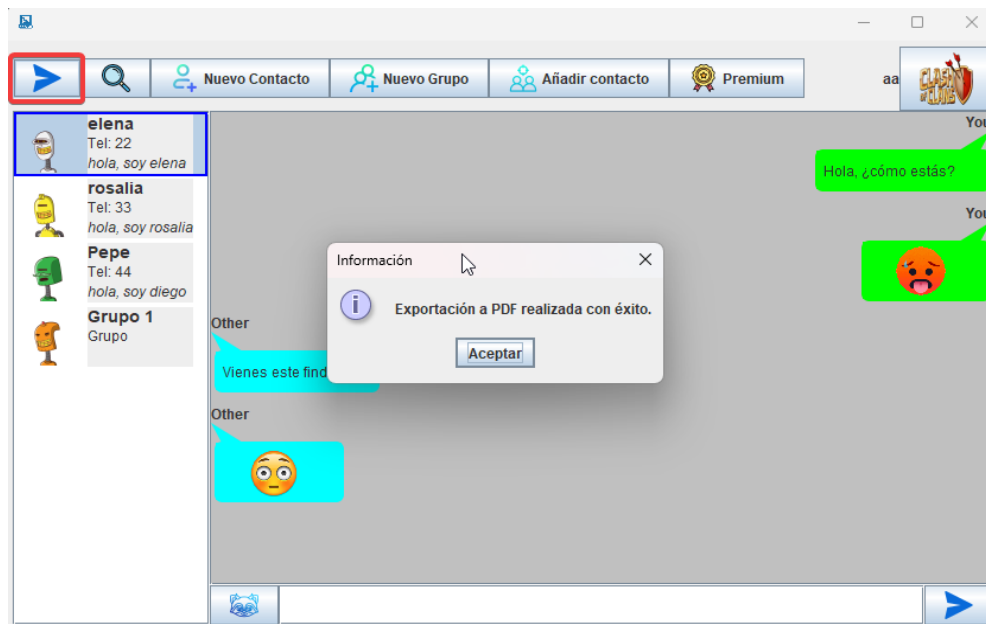


En este caso hemos buscado por “Hola”, buscará mensajes que contengan la cadena “Hola”

Por último, tenemos la opción de exportar todas nuestras conversaciones en PDF, para ello pulsaremos el botón de la izquierda, debemos tener en cuenta que para usarlo debemos ser usuario Premium sino nos aparecerá un mensaje como este:



Y si funciona correctamente nos saldrá una notificación, en la cual nos dirá que se ha podido realizar con éxito la exportación a PDF:



Y nos iremos a la carpeta de exportaciones y tendremos un PDF donde tendremos los usuarios, los mensajes de Contactos y los Mensajes que hayamos enviado a los grupos. A continuación, se deja una imagen de ejemplo de una pequeña parte del PDF para que se vea mejor:

Reporte de Usuarios y Grupos

Usuarios:

Nombre	Teléfono
elena	22
rosalia	33
Pepe	44

Mensajes con Contactos:

Conversación con: elena

Tipo	Fecha	Contenido
ENVIADO	2025-04-25T20:39:38.915950400	Hola, ¿cómo estás?
ENVIADO	2025-04-25T20:39:38.959016300	
RECIBIDO	2025-04-25T20:39:39.047367700	Vienes este finde?
RECIBIDO	2025-04-25T20:39:39.057393300	



7. Observaciones finales

El desarrollo de AppChat nos ha hecho aprender y desarrollar nuestras habilidades en el uso de Java, nos ha permitido poner en práctica muchos de los conocimientos adquiridos en la asignatura de Tecnologías del Desarrollo del Software. A lo largo del proyecto, nos hemos encargado tanto del diseño como de la implementación de una aplicación de mensajería funcional y bien estructurada, tomando como referencia la aplicación de uso cotidiano WhatsApp.

Durante el proyecto, hemos seguido principios de diseño sólidos como los establecidos en GRASP, lo que nos ha permitido una clara separación de responsabilidades entre vistas, lógica de negocio y persistencia. Gracias a esta organización, el sistema es fácil de mantener y extensible para futuras funcionalidades.

En cuanto a la arquitectura, hemos aplicado patrones de diseño como:

- **Strategy**, para la lógica de descuentos y filtros de búsqueda.
- **Método Factoría**, para la creación centralizada de adaptadores DAO.
- **Adaptador**, para desacoplar el controlador de la lógica de persistencia.

La elección y correcta aplicación de estos patrones nos ha permitido implementar funcionalidades avanzadas como:

- Registro e inicio de sesión de usuarios.
- Gestión de contactos individuales y grupos.
- Envío de mensajes con texto o emoticonos.
- Exportación de chats a PDF (funcionalidad premium).
- Búsqueda avanzada de mensajes mediante múltiples criterios.

Además, hemos implementado interfaces gráficas intuitivas con Swing, incluyendo renderizado personalizado de listas de contactos y mensajes, así como soporte visual para burbujas de chat, usando la librería aportada, y botones interactivos.

Como parte de la evolución del proyecto, también hemos resuelto retos técnicos como el renderizado de componentes interactivos en JList, o la integración con un sistema de persistencia embebido (H2) mediante adaptadores.

Finalmente, consideramos que AppChat refleja una aplicación robusta, coherente y completa que demuestra tanto el uso correcto de los principios teóricos de la ingeniería del software como la capacidad de llevarlos a la práctica en un entorno funcional. El resultado final ha sido muy satisfactorio, ya que hemos podido ver cómo todo el trabajo realizado se ha traducido en una aplicación que funciona bien y cumple con los requisitos pedidos. Creemos que refleja



todo lo que hemos aprendido durante la carrera hasta ahora, tanto a nivel técnico como en la manera de organizar y estructurar un proyecto real. Además, nos ha ayudado a ganar experiencia en el desarrollo de software modular y a entender mejor cómo aplicar en la práctica los conceptos que hemos visto en clase.

