

UNIVERSIDAD DE MURCIA
Facultad de Informática

PRÁCTICAS DE
Tecnologías de Desarrollo de Software

3º DE GRADO EN INGENIERÍA INFORMÁTICA

PROFESOR: FCO. JAVIER BERMUDEZ RUIZ

GRUPO 2.3

CURSO 2024/2025

José Salinas Pardo

48740555J

j.salinaspardo@um.es

Hugo Sánchez Martínez

24450997L

hugo.s.m@um.es

Contenidos

1. Introducción	1
2. Requisitos	1
3. Tecnologías	1
4. Requisitos y Modelado de Clases	2
4.1. <i>User Story Mapping</i>	2
4.2. Diagrama de secuencia: añadir miembro a grupo	2
4.3. Diagrama de clases	3
5. Arquitectura de la aplicación	5
5.1. Patrón MVC	5
5.2. Estructura general de la aplicación	6
5.3. Decisiones de diseño relevantes	6
6. Patrones de diseño	6
6.1. Patrones creacionales	6
6.2. Patrones estructurales	7
6.3. Patrones de comportamiento	8
6.4. Otros patrones de diseño	8
6.4.1. Data Access Object	8
6.4.2. Patrones usados indirectamente	9
7. Manual de usuario de AppChat	10
7.1. Inicio de sesión	10
7.1.1. Validación del inicio de sesión	10
7.2. Registro	11
7.2.1. Validación de campos en el registro	11
7.3. Vista principal	12
7.3.1. Búsqueda de mensajes	13
7.4. Gestión de contactos	13
7.4.1. Crear contacto	14
7.5. Perfil	14
7.6. Premium	15
7.6.1. Confirmación del pago	15
7.7. Exportación de mensajes en formato PDF	15
7.7.1. Ejemplo de documento generado tras exportar chats	16
8. Conclusiones personales y distribución del trabajo	17

Índice de figuras

1.	Diagrama de historias de usuario de AppChat	2
2.	Diagrama de secuencia simplificado para añadir miembro a grupo	2
3.	Modelo de clases de AppChat	3
4.	Patrón de arquitectura software MVC	5
5.	Jerarquía de paquetes de AppChat	6
6.	Modelo de <i>Singleton</i> en AppChat	6
7.	Ejemplo simplificado de <i>Facade</i> en el controlador de AppChat	7
8.	Modelo del patrón <i>Strategy</i> para descuentos en AppChat	8
9.	Modelo del patrón DAO junto con <i>Abstract Factory</i> y <i>Adapter</i> de AppChat	8
10.	Inicio de sesión en AppChat	10
11.	Error en el inicio de sesión	10
12.	Registro en AppChat	11
13.	Error en el registro	11
14.	Error en el registro	11
15.	Ventana principal de AppChat	12
16.	Búsqueda y filtrado de mensajes en AppChat	13
17.	Gestión de contactos de AppChat	13
18.	Ventana para añadir contactos	14
19.	Error al crear contacto	14
20.	Perfil de AppChat	14
21.	Actualización a AppChat <i>premium</i>	15
22.	Confirmación del pago de AppChat <i>premium</i>	15
23.	Ventana para exportar chats a formato PDF	15
24.	Confirmación del pago de AppChat <i>premium</i>	16

Resumen

Este documento especifica la implementación de las prácticas de la asignatura Tecnologías de Desarrollo de Software del tercer curso del grado en Ingeniería Informática de la Universidad de Murcia.

Dicha práctica consiste en el desarrollo de una aplicación de mensajería (*chatting*), **AppChat**, basada en aplicaciones ya existentes. El objetivo de este documento es describir en detalle el proceso de desarrollo de la aplicación, desde la planificación inicial y el análisis de requisitos, hasta la implementación y las pruebas finales.

1. Introducción

AppChat es una aplicación de escritorio de mensajería instantánea desarrollada en Java 8. Basada en soluciones reales como *Telegram* o *Whatsapp*, AppChat permite a los usuarios llevar una gestión simple de sus contactos, ofreciendo la posibilidad de crear grupos y enviar mensajes como si fuesen “*grupos de difusión*”.

Además, AppChat implementa funcionalidad extra para sus miembros “*premium*”. Estos pueden exportar sus conversaciones con sus contactos de una forma sencilla gracias a la librería *iText*.

2. Requisitos

AppChat requiere al menos tener instalada al menos la versión 8 de [Java SE](#) para ejecutar.

Es necesario tener instalado Maven para poder gestionar las dependencias del proyecto correctamente.

3. Tecnologías

Durante el desarrollo de AppChat se han usado las siguientes tecnologías:

- **Git**: Para el control de versiones y colaboración en el desarrollo.
- **Maven**: Utilizado como herramienta de gestión y construcción del proyecto.
- **Eclipse**: Se ha escodigo Eclipse como entorno de desarrollo (IDE).
- **PlantUML**: Usado para generar los diagramas UML del modelo.
- **Java Swing**: Biblioteca gráfica empleada para el desarrollo de la interfaz de usuario.

4. Requisitos y Modelado de Clases

4.1. User Story Mapping

Realizamos una **representación visual** de las funcionalidades de la aplicación desde la perspectiva de usuario. El eje horizontal representa el flujo de trabajo de un usuario en un orden lógico, mientras que el eje vertical organiza las historias en función de su importancia.

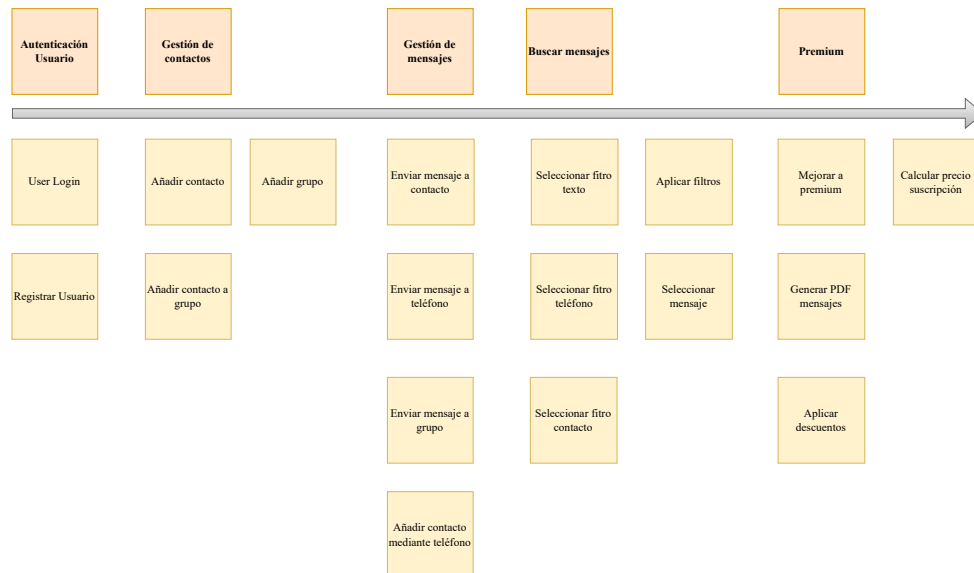


Figura 1: Diagrama de historias de usuario de AppChat

4.2. Diagrama de secuencia: añadir miembro a grupo

La funcionalidad tomada de ejemplo es la de crear un grupo, pero es válida ya que esta añade de forma secuencial varios miembros al grupo.

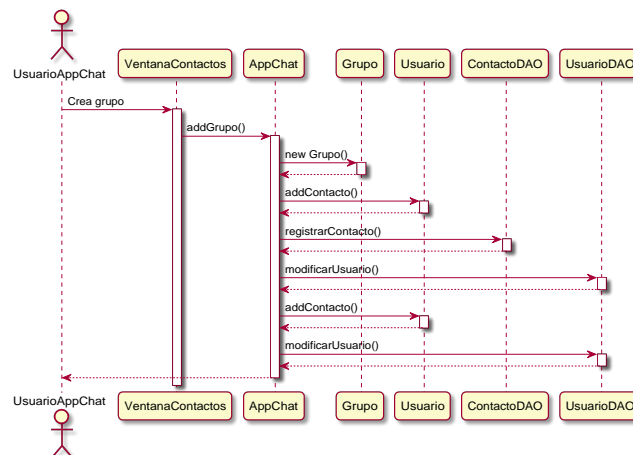


Figura 2: Diagrama de secuencia simplificado para añadir miembro a grupo

4.3. Diagrama de clases

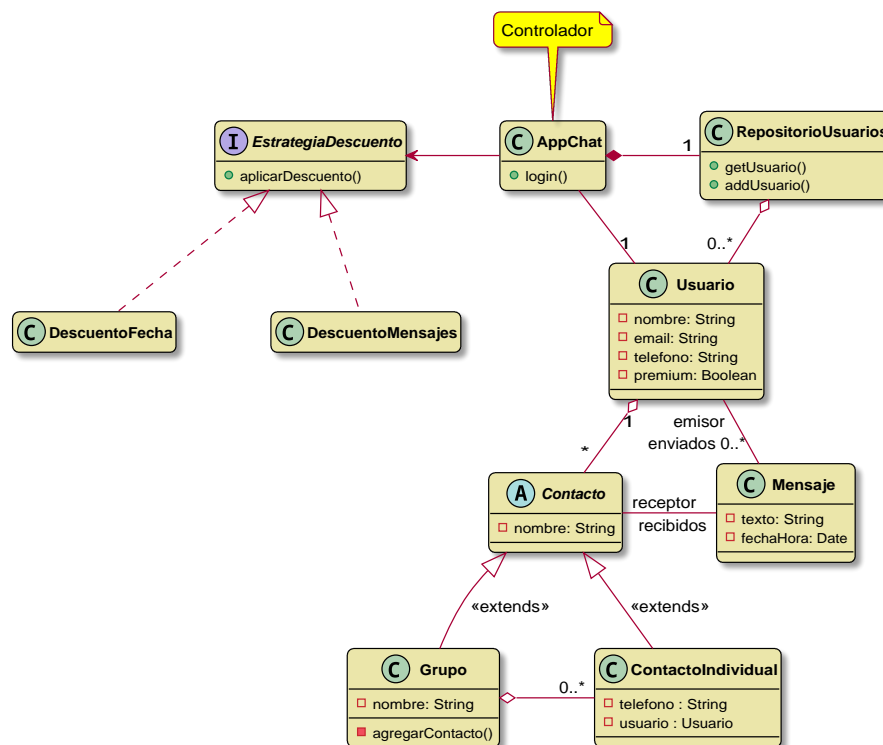


Figura 3: Modelo de clases de AppChat

Se ha optado por seguir la recomendación de los profesores de la asignatura, con ligeros cambios. La figura 3 no muestra todos los atributos y métodos, solo los más relevantes.

- **AppChat** : Siguiendo el principio MVC que se explicará más adelante, esta clase actúa de «controladora» de la aplicación.
- **RepositorioUsuarios** : Contiene una colección de los usuarios gestionados. Para la lógica de la aplicación, además de las entidades persistentes en la base de datos, es necesario llevar un control de todos los usuarios. Es invocada por el controlador con frecuencia, por ejemplo, para cargar los usuarios al iniciar la aplicación.
- **Usuario** : Representa al usuario de AppChat. Incluye operaciones para interactuar con sus contactos.
- **Contacto** : Clase abstracta que representa una forma de contacto con otro usuario. TODO.
 - **ContactoIndividual** : Representa el contacto íntegro de la aplicación. Está asociado a un usuario de AppChat y tiene asignado un nombre de contacto.
 - **Grupo** : Clase que modela un chat formado por un conjunto de contactos. Tiene un único administrador.
- **Mensaje** : Clase que representa todo lo relacionado con un mensaje en la aplicación. Tiene como emisor un objeto **Usuario** y como receptor un objeto **Contacto**, que puede ser una persona o un grupo.
- **EstrategiaDescuento** : Interfaz necesaria para implementar diferentes descuentos usando el patrón de diseño “strategy”.

- `DescuentoFecha` : Estrategia de descuento que está disponible para el usuario si su fecha de nacimiento está entre un intervalo establecido por el sistema.
- `DescuentoMensajes` : Estrategia de descuento que aplica un porcentaje al precio base si el usuario supera un límite establecido de mensajes enviados en la aplicación.

También se han diseñado otras clases como:

- `App` : Lanzador de la aplicación.
- `CriteriosBusqueda`
- `BuscadosMensajes`

5. Arquitectura de la aplicación

5.1. Patrón MVC

AppChat sigue el patrón de arquitectura de software Modelo-Vista-Controlador. Este patrón nos permite separar los datos y la lógica del negocio de una aplicación de la interfaz que ve el usuario, conéctandolos a través de un controlador, permitiendo así tres componentes distintos: [1]

- **Vista (View):** Representa la interfaz de usuario y todas las herramientas con las cuales el usuario hace uso del programa.
- **Modelo (Model):** Es dónde está toda la lógica del negocio, la representación de todo el sistema, incluida la interacción con una base de datos, si es que el programa así lo requiere.
- **Controlador (Controller):** Este componente es el que responde a la interacción (eventos) que realiza el usuario en la interfaz y realiza las peticiones al modelo para luego pasarlas a la vista. Aún no está del todo claro, pero con el siguiente ejemplo lo comprenderemos mejor.

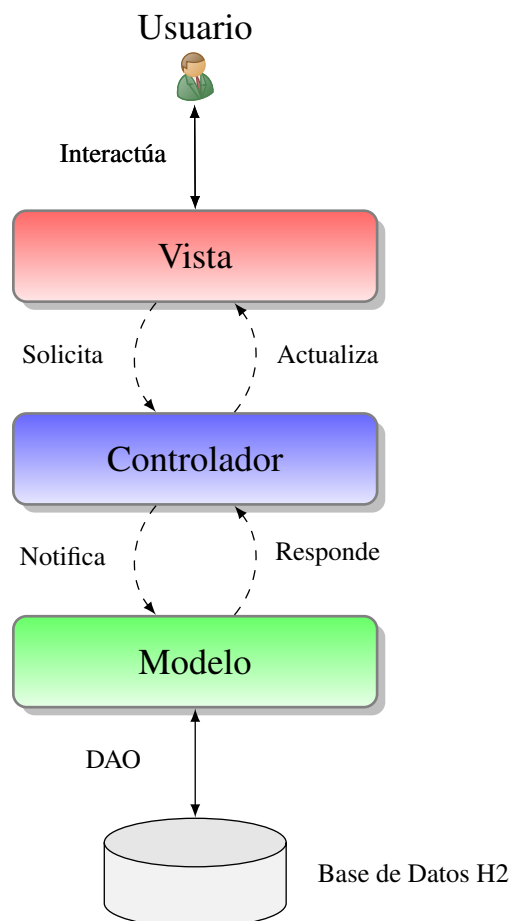


Figura 4: Patrón de arquitectura software MVC

5.2. Estructura general de la aplicación

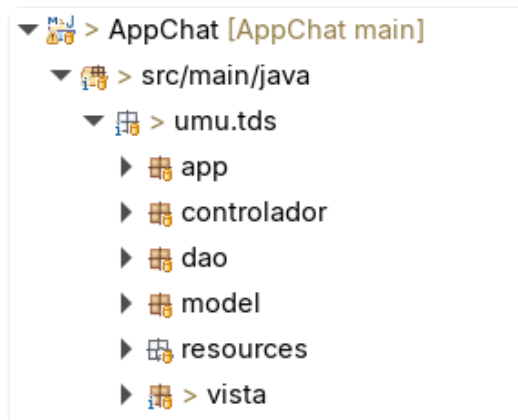


Figura 5: Jerarquía de paquetes de AppChat

La aplicación se estructura en 5 paquetes diferentes, cada uno con una funcionalidad específica. Como se ha comentado, los paquetes `model`, `controlador` y `vista`, forman parte del patrón MVC. Por su parte,

5.3. Decisiones de diseño relevantes

6. Patrones de diseño

A lo largo del desarrollo de la aplicación, se han adoptado los principales patrones de diseño vistos en clase y se han implementado directamente en el código.

6.1. Patrones creacionales

Según *Gamma et al.* [2] el objetivo principal de los patrones de diseño creacionales es el de **abstraer el proceso de instanciación y creación de objetos**. Los patrones creacionales permiten además encapsular el conocimiento sobre las clases concretas que utiliza el sistema, así como ocultar el proceso de creación y ensamblación de instancias por parte de las clases.

Singleton

Utilizamos el patrón *singleton* para aquellas clases de las que nos interesa tener una única instancia y permitir un acceso global a ella. En AppChat, siguen este patrón la clase controlador `AppChat`, el repositorio `RepositorioUsuarios`, la clase “factoría” `DAOFactory` y el resto de clases “adaptador” del paquete `umu.tds.dao.tds`.

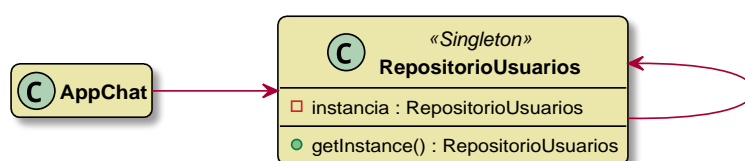


Figura 6: Modelo de *Singleton* en AppChat

Factoría abstracta

Este patrón se usa para las clases relacionadas con el ámbito de la persistencia en la aplicación. Se define una clase abstracta `DAOFactory` implementada como singleton y con métodos para obtener los distintos adaptadores disponibles.

6.2. Patrones estructurales

Adaptador

El patrón *adapter* o adaptador permite que clases con interfaces incompatibles colaboren entre sí sin necesidad de modificar su código fuente. Usamos este patrón en las clases del paquete `umu.tds.dao.tds` para adaptar la funcionalidad de la clase `ServicioPersistencia` a las interfaces DAO definidas en `umu.tds.dao`. En la figura 9 se puede ver con más claridad la interrelación entre las clases.

Facade

El patrón *facade* o fachada se usa para proporcionar una interfaz que actúa como punto de acceso simplificado a un subsistema complejo. Su objetivo principal es ocultar la complejidad interna de varias clases colaboradoras y ofrecer una interfaz más clara y coherente. [3]

Este patrón se usa en la clase controlador `AppChat`. Como se puede ver en el diagrama de secuencia 7, para operaciones como “añadir contacto” es necesario una interfaz que oculte las complejas operaciones que conlleva la función.

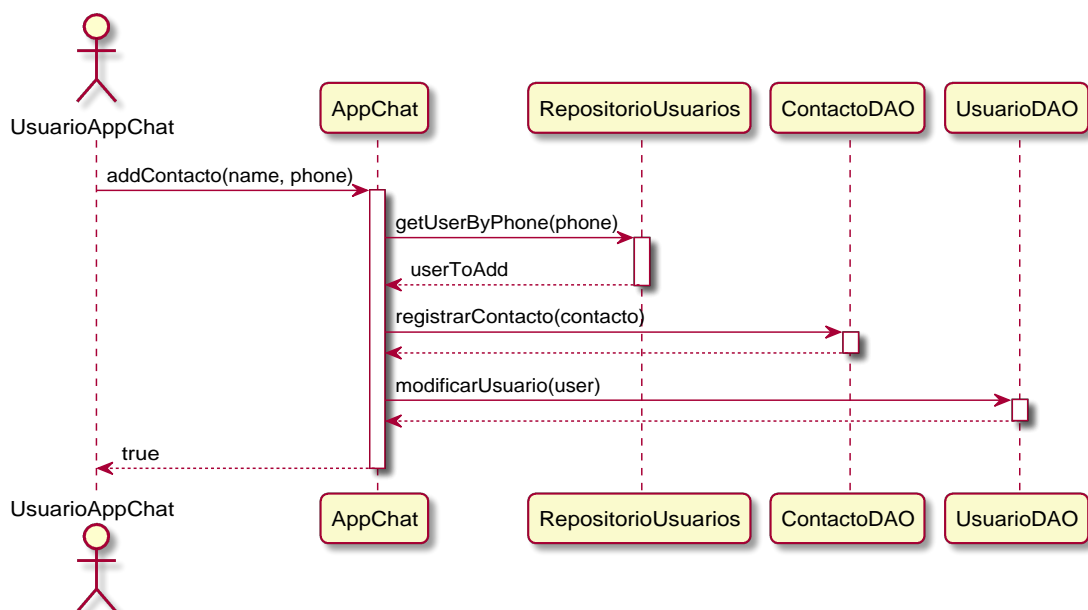


Figura 7: Ejemplo simplificado de *Facade* en el controlador de AppChat

6.3. Patrones de comportamiento

Strategy

Implementamos patrón de diseño *Strategy* o Estrategia para diseñar los descuentos en AppChat. Nos permite definir una familia de algoritmos y encapsular cada uno de ellos. En el futuro, se puede implementar un nuevo descuento de forma sencilla, usando la interfaz definida.

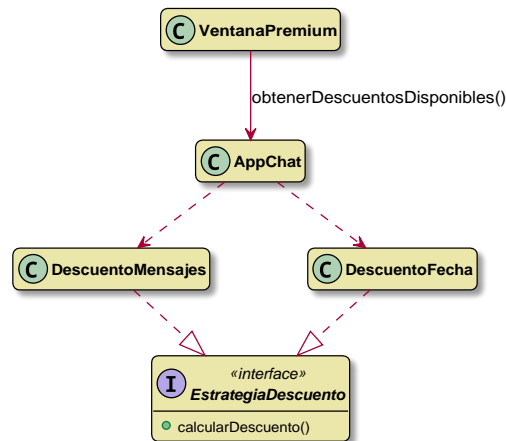


Figura 8: Modelo del patrón *Strategy* para descuentos en AppChat

6.4. Otros patrones de diseño

6.4.1. Data Access Object

El patrón Data Access Object o DAO nos permite separar por completo la lógica de negocio de la lógica para acceder a los datos. De esta forma, las clases DAO proporcionarán los métodos necesarios para las operaciones de persistencia de los objetos, como guardar, borrar, editar o recuperar entidades. este patrón se usa junto factoría abstracta y adaptador para proporcionar una interfaz clara y desacoplar los paquetes.

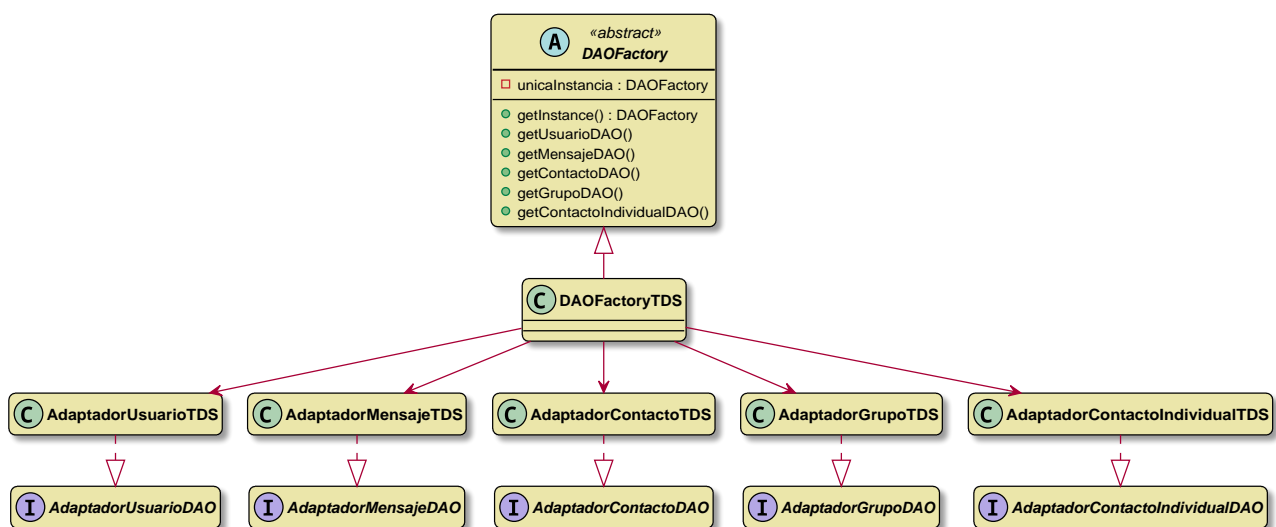


Figura 9: Modelo del patrón DAO junto con *Abstract Factory* y *Adapter* de AppChat

6.4.2. Patrones usando indirectamente

En el desarrollo de AppChat, se han usado numerosas clases que implementan patrones de diseño en su código. Destacan algunos como:

- **Singleton, Facade, y Abstract Factory**: la librería DriverPersistencia proporcionada por los profesores usa proporciona una interfaz clara y un método de acceso global a la clase `FactoriaServicioPersistencia`.
- **Composite**: las clases `JPanel`, `JFrame` y `Container` de Java AWT y Swing implementan este patrón para poder tratar objetos individuales y grupos de objetos de manera uniforme.
- **Observer**: cuando usamos, por ejemplo, “*action listeners*” en los botones de las ventanas.
- **Iterator**: en las colecciones.

7. Manual de usuario de AppChat

7.1. Inicio de sesión



Figura 10: Inicio de sesión en AppChat

7.1.1. Validación del inicio de sesión

La interfaz gráfica ejecuta métodos y se comunica con el controlador para verificar el inicio de sesión. Si un usuario introduce un número no registrado o una contraseña incorrecta, el sistema mostrará un diálogo de error:



Figura 11: Error en el inicio de sesión

7.2. Registro



The screenshot shows a window titled "Registrarse en AppChat" with a close button (x) in the top right corner. At the top center is the AppChat logo, which consists of a blue speech bubble with three white dots and the word "appchat" below it. Below the logo is a section titled "Create account" containing several input fields: "Phone number", "First name", "Last name", "Password", "Confirm password", "Date" (with a calendar icon), "Profile picture" (with a "Generate" button and "No image" text below it), and "Greeting" (with a text area and scrollbars). At the bottom left is a "Cancel" button and at the bottom right is a "Confirm" button.

Figura 12: Registro en AppChat

7.2.1. Validación de campos en el registro

La interfaz gráfica también incorpora métodos para validar los campos antes de registrar a un usuario. Por ejemplo, si se intenta usar un número de teléfono ya registrado, el sistema mostrará un error al intentar crear la cuenta.

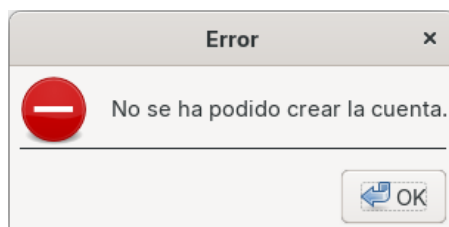


Figura 13: Error en el registro

Asimismo, si el usuario no introduce la misma contraseña los campos “Contraseña” y “Confirmar contraseña”:



Figura 14: Error en el registro

7.3. Vista principal

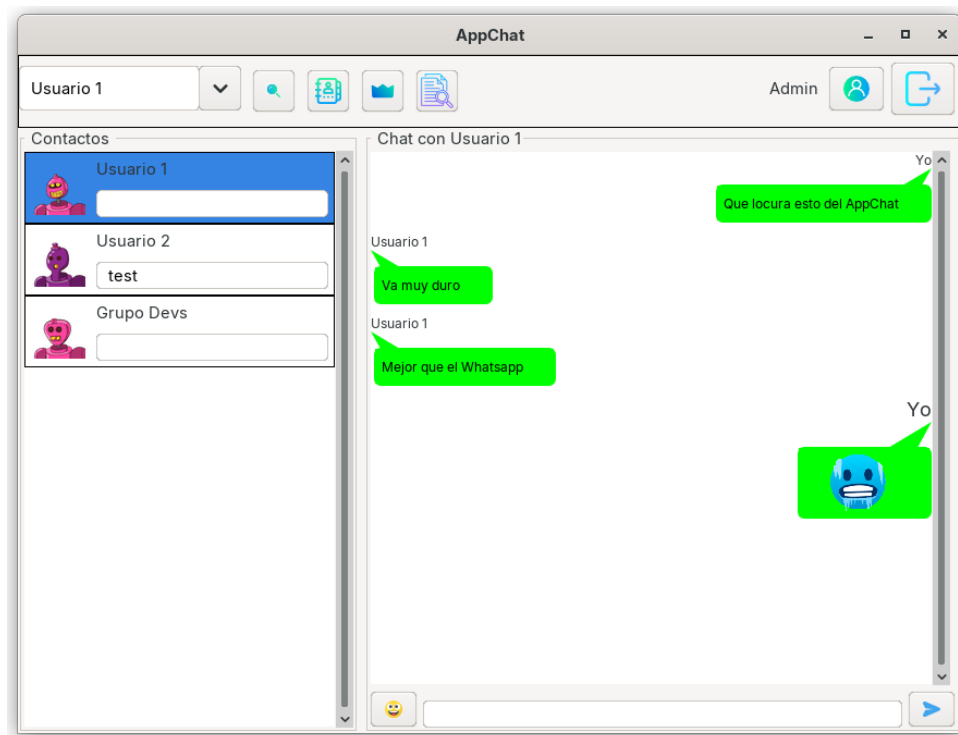


Figura 15: Ventana principal de AppChat

7.3.1. Búsqueda de mensajes

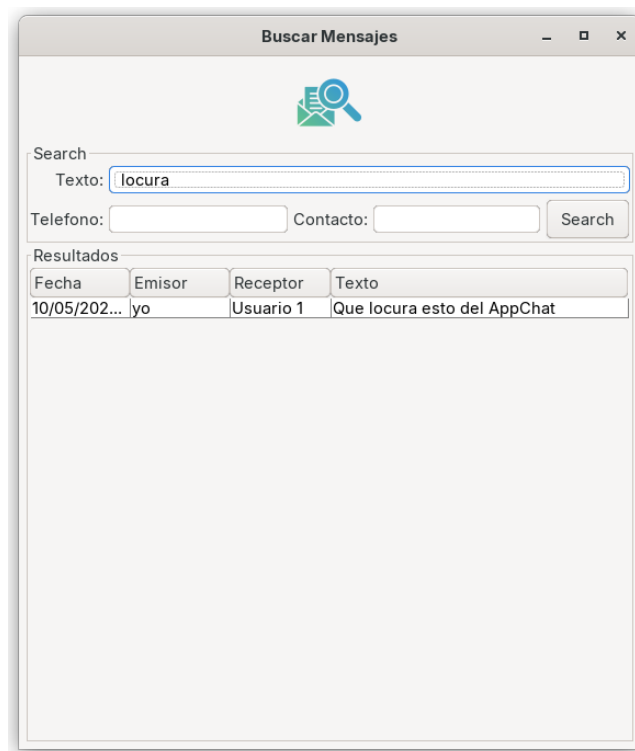


Figura 16: Búsqueda y filtrado de mensajes en AppChat

7.4. Gestión de contactos



Figura 17: Gestión de contactos de AppChat

7.4.1. Crear contacto

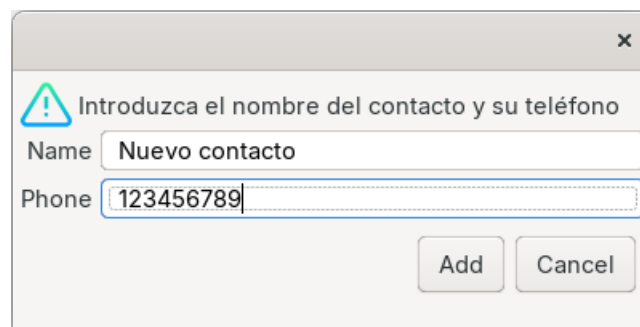
A dialog box titled "Introduzca el nombre del contacto y su teléfono" with a warning icon. It contains two input fields: "Name" with the text "Nuevo contacto" and "Phone" with the text "123456789". At the bottom right are "Add" and "Cancel" buttons.

Figura 18: Ventana para añadir contactos

Error al crear contacto

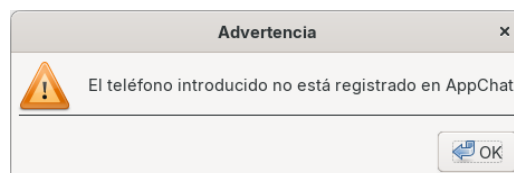


Figura 19: Error al crear contacto

7.5. Perfil

A form titled "Editar Perfil" (Edit Profile). It is divided into two main sections. The left section, "Información del Usuario" (User Information), contains fields for "Nombre compl..." (display name) with the value "Admin", "Teléfono:" (phone) with the value "1", "Fecha nacimie..." (birth date) with the value "15/05/2025", and a "Descripción:" (description) field. Below these fields is a note: "* Esta información no es editable en est...". The right section, "Imagen de Perfil" (Profile Image), displays a circular profile picture of a person. At the bottom of the form are three buttons: "Seleccionar imagen" (Select image), "Guardar" (Save), and "Cancelar" (Cancel).

Figura 20: Perfil de AppChat

7.6. Premium

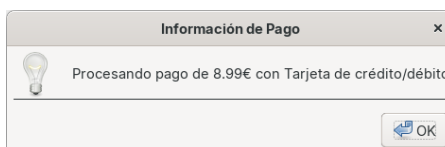


The screenshot shows a window titled "AppChat Premium" with a close button (x). Inside, there is the AppChat logo (a blue speech bubble with a yellow star) and the text "appchat". Below this, a section titled "Suscripción AppChat Premium" contains the following information:

- Precio de suscripción: 9.99€
- Descuento disponible: Promoción primavera (10%) (selected from a dropdown menu)
- Precio final: 8.99€
- Seleccione método de pago: Tarjeta de crédito/débito (selected from a dropdown menu)
- Form fields for: Número de tarjeta, Titular de la tarjeta, Fecha de expiración (MM/AA), and CVV.
- Buttons: "Cancelar" and "Confirmar Pago".

Figura 21: Actualización a AppChat *premium*

7.6.1. Confirmación del pago



The screenshot shows a window titled "Información de Pago" with a close button (x). Inside, there is a lightbulb icon and the text "Procesando pago de 8.99€ con Tarjeta de crédito/débito". At the bottom right, there is an "OK" button with a blue arrow icon.

Figura 22: Confirmación del pago de AppChat *premium*

7.7. Exportación de mensajes en formato PDF

Para esta funcionalidad, se ha implementado también la posibilidad de poder exportar los mensajes con todos los contactos.



The screenshot shows a window titled "Exportar Conversación a PDF" with a close button (x). Inside, there is a section titled "Exportar Conversación" with the following elements:

- Selección de contacto: "Todos los contactos" (selected from a dropdown menu).
- Guardar en: A text input field and a button labeled "Seleccionar ubicación...".
- Buttons: "Cancelar" and "Exportar".

Figura 23: Ventana para exportar chats a formato PDF

7.7.1. Ejemplo de documento generado tras exportar chats



appchat

INFORME DE CONVERSACIONES

Fecha de generación: 11/05/2025 20:33:21
Generado por: Admin (ID: 1)

RESUMEN DE MENSAJES
Total de contactos: 3

Contacto: Usuario 2

ID	Emisor	Fecha y Hora	Mensaje
113	Admin	2025-05-11T19:58:34	test

Contacto: Usuario 1

ID	Emisor	Fecha y Hora	Mensaje
43	Admin	2025-05-10T17:30:19	Que locura esto del AppChat
101	Usuario1	2025-05-11T18:29	Va muy duro
107	Usuario1	2025-05-11T18:29:12	Mejor que el Whatsapp

Contacto: Grupo Devs
No hay mensajes con este contacto.

Documento generado automáticamente por AppChat con iText.

Figura 24: Confirmación del pago de AppChat *premium*

8. Conclusiones personales y distribución del trabajo

Este proyecto ha sido el más extenso hasta el momento del grado en Ingeniería Informática también el primero en el que hemos desarrollado una aplicación gráfica de escritorio.

Ha sentado las bases sobre conceptos como la lógica de negocio (*backend*) y la interfaz de usuario (*frontend*). La interfaz proporcionada para la persistencia de datos también ha ayudado a comprender cómo se almacenan los objetos Java de forma persistence, y ha servido como una guía para más adelante usar JPA o Jakarta.

Al ser el primer proyecto de esta índole, las prioridades y ejecución del proyecto eran desconocidas. Si intentamos relacionar la práctica con otras asignaturas del grado, encontramos una gran similitud Gestión de Proyectos de Desarrollo de Software o Procesos de Desarrollo de Software. Hemos ejecutado el desarrollo de un proyecto de software y hemos aprendido la importancia del análisis de requisitos y del diseño de modelos.

Usar librerías externas nos ha mostrado la relevancia que tienen los patrones de diseño en el desarrollo de aplicaciones en la actualidad. Cada uno tiene un objetivo, y su uso resulta imprescindible en proyectos extensos.

Actividad	José Salinas	Hugo Sánchez
<i>Historias de usuario</i>	4 h	3 h
<i>Diseño del modelo</i>	4 h	2 h
<i>Implementación de la lógica</i>	65 h	90 h
<i>Diseño de la interfaz gráfica</i>	90 h	65 h
<i>Revisión y corrección de errores</i>	20 h	15 h
<i>Mantenimiento del repositorio</i>	2 h	2 h
<i>Documentación</i>	4 h	9 h
Total	189 horas	186 horas

Cuadro 1: Distribución de horas entre los miembros

Referencias

- [1] MDN Web Docs. Mvc (model-view-controller). <https://developer.mozilla.org/en-US/docs/Glossary/MVC>, 2024. Accessed: 2025-03-11.
- [2] Ralph Johnson John M. Vlissides Erich Gamma, Richard Helm. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, illustrated edition edition, 1994.
- [3] Alexander Shvets. Facade - design patterns. <https://refactoring.guru/design-patterns/facade>, 2023.