



---

# PRACTICA - TDS

---

Jesús Marí Alcaraz 17471130P – Arturo Lorenzo Hernández 48693088H



11 DE ENERO DE 2020

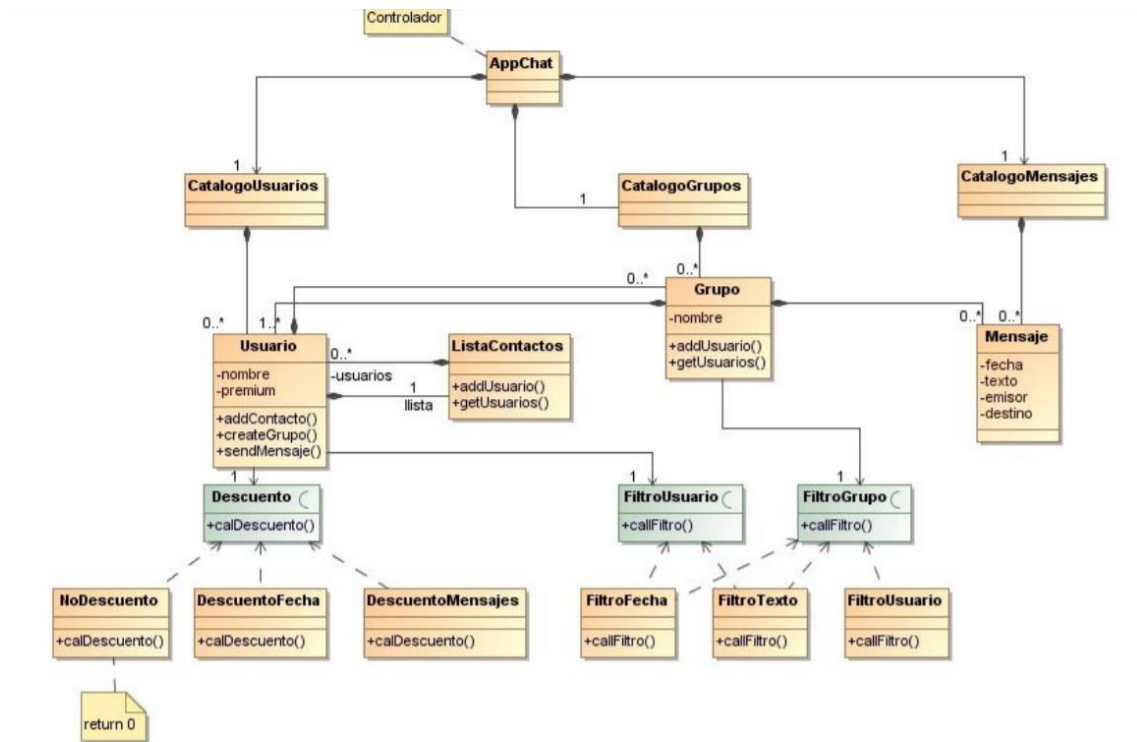
TDS

## Contenido

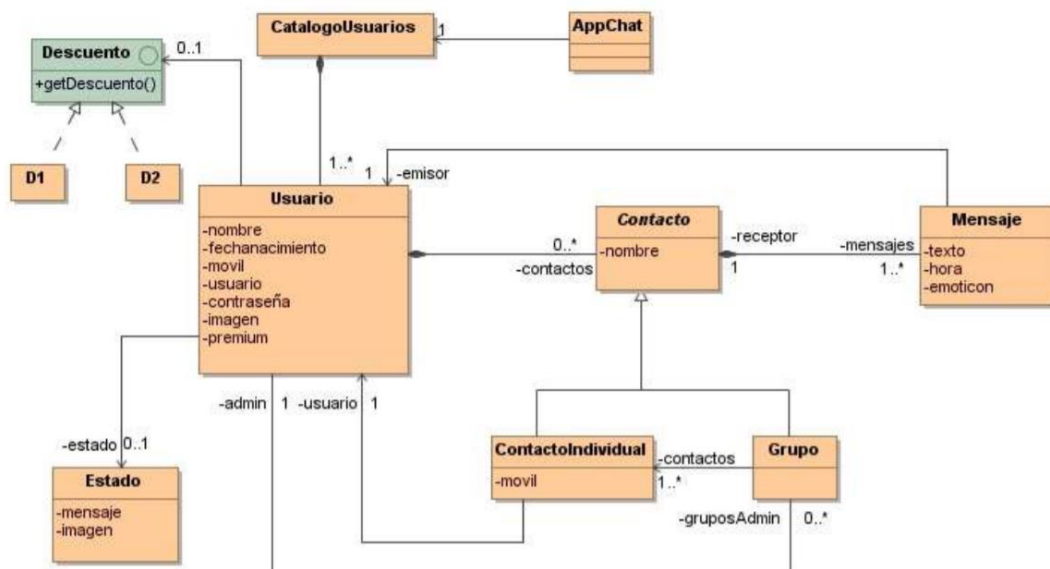
Diagrama de Clases del Dominio .....	2
Diagrama de Secuencia .....	3
Explicación Arquitectura .....	3
Explicación Patrones .....	4
Patrón DAO .....	4
Patrón Composite .....	4
Componentes .....	4
JCalendar .....	4
JUnit .....	4
JavaBeanCargadorMensajes .....	4
Luz .....	4
Tests .....	4
Manual de Usuario .....	5
Observaciones Finales .....	6

## Diagrama de Clases del Dominio

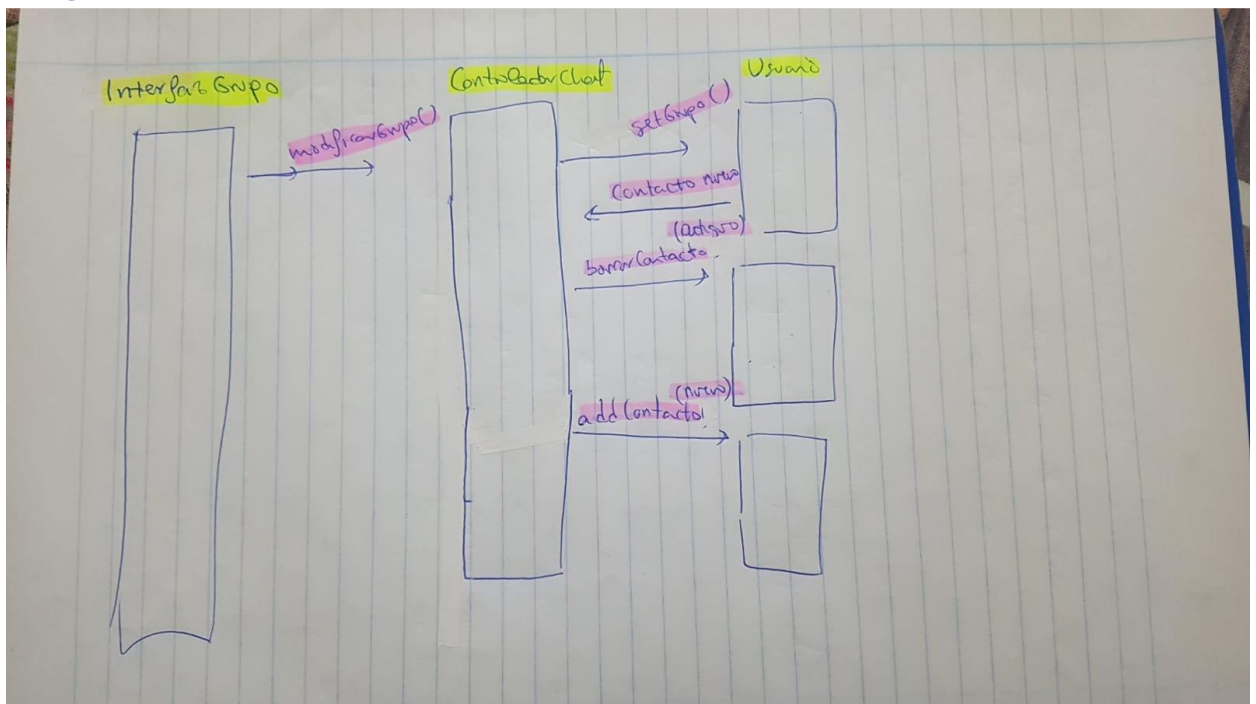
Aquí esta nuestro primer diagrama, el cual no hemos seguido ya que estaba mal:



Entonces hemos implantado la aplicación el base al diagrama dado:



## Diagrama de Secuencia



## Explicación Arquitectura

Hemos utilizado el principio de separación en tres capas. (Dominio, Vista, Controlador), para mejorar el extensibilidad y legibilidad del código de la aplicación.

En el dominio tenemos las clases de Usuario, Contacto (Abstracta), ContactoIndividual, Grupo, Mensajes y Catalogo de Usuarios, las cuáles implementan la lógica del negocio y las clases principales de la aplicación.

El Controlador se encarga de ser intermediario entre la capa de la vista y capa del dominio, sin saber nada de la interfaz, esto lo consigue a través de distintas llamadas a funciones de objetos del dominio y desarrollando cierta parte de la lógica de la aplicación.

La capa de la vista esta representada con una interfaz principal llamada MainView de la que derivan todas las demás interfaces que implementan las distintas funcionalidades de la aplicación, estas vistas si que tienen conocimiento de la lógica del negocio y por ello hacen llamadas a los métodos del controlador, lo que no se implementa en estas interfaces es el desarrollo de la lógica del negocio.

También tenemos separado el servicio de persistencia donde hemos utilizado un patrón DAO para crear una instancia de H2. (Ya que solo hemos implementado H2 y no MySQL), dicho patrón nos da un acceso a través de un objeto factoría abstracta a las distintas familias de adaptadores en los cuáles se implementan los distintos objetos que persistimos en la base de datos. Por sólo utilizar el servicio de H2 tenemos un único tipo de adaptadores dao.

## Explicación Patrones

### Patrón DAO

Hemos utilizado el patrón DAO para conseguir que la aplicación sea independiente del sistema de almacenamiento, aunque solo hemos implementado H2 y no mysql. Hemos utilizado una factoría abstracta para crear la instancia de H2. También hemos utilizado el patrón adaptador que está incluido en el patrón DAO, para poder realizar las operaciones CRUD necesarias para poder persistir los objetos pertinentes. Este patrón nos ayuda a conseguir a través de la API de H2 crear la funcionalidad requerida.

### Patrón Composite

El patrón composite lo hemos utilizado en la creación de grupos y usuarios, para que un grupo pueda tener varios usuarios, pero con la particularidad de que los grupos no pueden tener otros grupos.

### Patrón Estrategia

Hemos empleado el patrón estrategia para aplicar una política de descuentos en la aplicación a la hora de que un usuario se hace premium en la cuál se distinguen dos políticas: una de descuento fijo del 10% sobre el total (10 euros) y otra de descuento joven por nacer en el año 1998 o más tarde del 30% sobre el total. Este patrón nos permite definir familias de algoritmos de forma que el objeto que implementa el descuento en nuestro caso el usuario es independiente de la jerarquía de descuentos.

## Componentes

### JCalendar

El componente JCalendar lo hemos usado para introducir la fecha de registro y las búsquedas de los mensajes, en los cuáles hacemos búsquedas de mensajes entre dos fechas dadas.

### Junit

Junit lo hemos utilizado para realizar los test de la aplicación lo explicamos más adelante en el apartado de Tests.

### JavaBeanCargadorMensajes

Un componente para cargar distintos mensajes, donde especificas el fichero con la conversación parseada para un sistema operativo específico.

### Luz

Luz componente otorgado por la universidad, que hemos utilizado para cargar conversaciones de distintos sistemas operativo, cuando le das al botón de cargar en la interfaz del perfil de usuario el componente hace uso del javabeen cargador de mensajes.

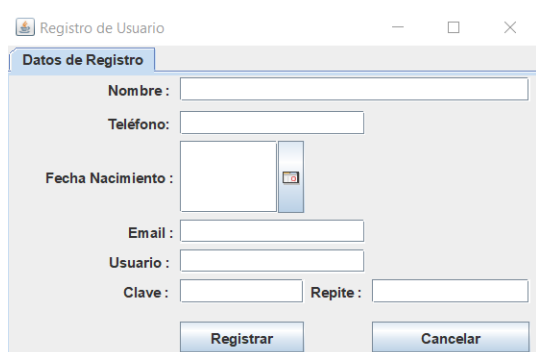
## Tests

Hemos creado 5 tipos de test.

- Que comprueba si se crea bien el login y la contraseña de usuario.
- Si un grupo se crea con el número de participantes correctos.
- Si un contacto tiene el número de referencia correcto a un usuario.
- Que un usuario mande correctamente un mensaje a un usuario
- Y que un usuario añada correctamente un contacto.

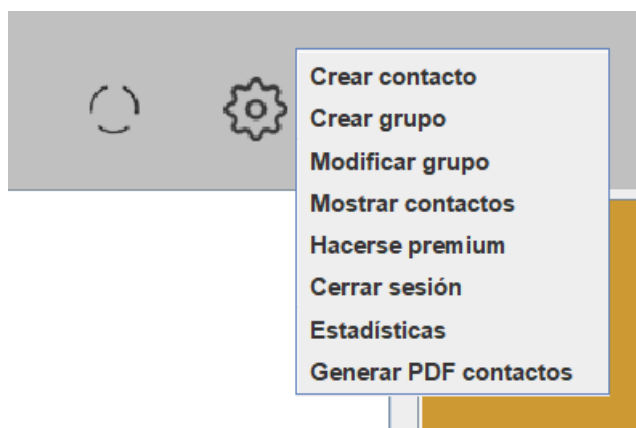
## Manual de Usuario

Para ejecutar la aplicación hacemos click en el archivo jar proporcionado "App.jar" que iniciará la aplicación, para acceder a ella tendremos que loguearnos con una cuenta creada, o registrar una nueva cuenta:



A screenshot of a Java Swing window titled "Registro de Usuario". It contains a tab labeled "Datos de Registro". The form has the following fields: "Nombre:" (text), "Teléfono:" (text), "Fecha Nacimiento:" (calendar icon), "Email:" (text), "Usuario:" (text), "Clave:" (password), and "Repite:" (password). At the bottom are two buttons: "Registrar" and "Cancelar".

Una vez dentro de la aplicación podremos crear contactos, si ya tenemos contactos podremos crear grupos también, borrar contactos, modificar un grupo si eres el administrador, convertirse en premium, cerrar la sesión, crear las estadísticas pedidas para la aplicación y la generación de PDFs.



Para hablar con un contacto primero tendremos que seleccionarlo de la lista de contactos. Si ya hemos hablado con el en la parte izquierda nos saldrán los últimos contactos con los que hemos hablado.



Una vez seleccionado un contacto podemos borrar mensajes o buscar mensajes dentro de la conversación, si es un grupo podemos buscar por usuario también.



**BUSQUEDA DE MENSAJES**

Mensaje

Nombre Usuario

Fecha1

Fecha2

Buscar

**MENSAJES ENCONTRADOS**

También podemos importar mensajes de otros sistemas operativos haciendo uso del botón:

Input

Introduce modelo de mensajes que quieres importar (IOS,Android1,Android2)

OK Cancel

Introduciendo el tipo de sistema operativo del que proceden esos mensajes.

## Observaciones Finales

Tiempo consumido:

40 horas en la creación de la Interfaz.

10 horas creación del servicio de persistencia.

10 horas creación del controlador.

20 horas creación de clases del dominio

10 horas creación y adaptación de componentes.

10 horas corrigiendo y solucionando problemas.

Total, tiempo consumido aproximado: 100 horas.