# “My Travel Wallet”

## Proyecto de Línea 3 - Elaboración de una Aplicación para Dispositivo Móvil

## Ciclo Formativo de Grado Superior de Desarrollo De Aplicaciones Multimedia

**Sergio Penadés Escrivá**

**Alfonso García**

# Curso 2023-2024

## ÍNDICE

# 1. Resumen

El objetivo del desarrollo de esta aplicación, es debido a la necesidad de poder operar con pagos comunes, por ejemplo en viajes, de una manera muy intuitiva y sencilla.
Las aplicaciones existentes hoy en día, están llenas de opciones, que a nivel práctico, son prescindibles, y con esta aplicación se intenta que la experiencia de usuario sea lo más agradable y sencilla posible.
Para el desarrollo, se ha elegido el “modelo por etapas”. Este método implica que la aplicación en todo momento es funcional, y ha ido evolucionando poco a poco.
Tenemos una parte gráfica, muy intuitiva, y que de una simple mirada, se ven los datos principales. Existe la posibilidad de editar casi todos los objetos que hemos introducido, y esto le da mucha flexibilidad.
La aplicación, nos expone en cada momento un detalle importante, saber quién debería pagar próximamente para igualar los saldos deudores, o cual es el importe total de cada “Wallet” (del inglés, monedero, cartera, billetera). De forma más precisa, desde la pantalla de “Saldar Deudas”, tenemos la información sobre que ha pagado cada miembro, y cuanto adeuda, estos datos ofrecen ese punto de información necesaria y concisa. El punto más importante de la aplicación, se encuentra en esta pantalla de “Saldar Deudas”. El código que materializa esta pantalla es un código que realiza todo los cálculos necesarios para que la operación sea precisa, ofreciendo una vez más, una información pertinente y concisa, donde, de un vistazo, encontramos la solución a las deudas pendientes. En conclusión, ha sido un trabajo desafiante, sobre todo a nivel de cálculo de las deudas, y también en manejo de persistencia de datos, que han ido formando retos, a cual más interesante, y que gracias a todo lo aprendido durante el curso, y con ayuda de internet, y la lectura de algunos libros, se han podido ir superando poco a poco, dando pie a la incorporación de opciones, que en un principio no tenía presentes implementar.

*Palabras clave:* aplicación, deuda, cálculo, compensación, viaje, gasto.

## 2. Abstract

The aim of developing this App is because of the need being able to deal with common payments, for instance,when in travelling, in a very intuitive and simple way.
The nowadays existing Apps are full of options that, on a practical level, are dispensable, and with this App we try to make the user‘s experience as pleasant and simple as possible.
For the development, the model has been chosen “in stages”. This method implies that the App is functional at all times, and has been evolving bit by bit.
We have a graphic part, very intuitive, and at a glance, you can see the main data. There is the option of editing almost all the objects that we have introduced, and this provides full flexibility.
The App does not present an excessive volume of information, but it automatically shows us at all times the most important details, such as knowing who should pay next, to equalize the debtor balances, or how much i the total amount of each “Wallet”. In a more precise way, from the “Pay Debts” screen we have the information about how much each member has paid and how much they owe, which gives them that point of required and concise information. The most important thing of the App is found in this “Settle Debts” screen. The code that materializes this screen is a code that performs all the necessary calculations so the operation is accurate, offering once again, pertinent and concise information, where, at a glance, we find the solution to outstanding debts. In conclusion, it has been a challenging job, especially at the level of calculating debts, and also in managing data persistence, which have created challenges, each one more interesting than the previous one, and thanks to everything learned during the course, and with the help of the Internet, and reading some books, i have been able to be overcome little by little, resulting in the incorporation of options, which at first I did not have in mind to implement.

*Keywords*: app, debt, calculation, compensation, travel, spending.

# 3. Introducción

Ante varios gastos entre distintas personas, si son un pequeño número de transacciones, es muy fácil “cuadrar las cuentas”, pero ¿qué sucede cuando el número de transacciones son elevadas o son muchos los participantes?

Pues bien, si tenemos en mente tanto una cuenta de compañeros de piso, o en pareja, o algo más complicado, como un viaje de varias personas, los gastos son siempre un motivo de preocupación.

Si bien existen aplicaciones de esta temática, como por ejemplo GroupXpense, ExpenseCount, resultan aplicaciones con opciones que muchas veces no se utilizan, ante esto, la valoración de realizar una aplicación que fuera, austera, a la par que muy práctica, es lo que ha llevado al desarrollo de “My Travel Wallet”, mi billetera de viaje.

Si tomamos como ejemplo un viaje, en él se realizan pagos comunes, tales como comidas, bebidas, gastos de combustible, entre otros. Una opción muy extendida es que el pago lo efectúe uno de los participantes, que actuaría en forma de “banca”, de forma que todos los integrantes del grupo irían depositando la misma cantidad de dinero en dicho fondo común o “banca”. Esto, hoy en día resulta muy engorroso, sobre todo para el que se defina como “banca”.

Con el fin de despreocuparse de dichas finanzas y de los problemas que puedan surgir, esta aplicación ayudará en dichas operaciones conjugando las nuevas tecnologías, e impulsando la banca electrónica, puesto que al finalizar la existencia del “Wallet”, se pueden realizar las transferencias de los importes sugeridos mediante “Bizum”, así pues, se minimiza al máximo el intercambio de efectivo entre los participantes. Y evitamos la problemática de tener que llevar billetes y monedas para poder realizar los pagos.

Hoy en día y tras la pandemia del COVID-19, y por miedo al contagio, se tiende a las compras Online, y al pago mediante transacciones digitales, así pues el 47% de los

españoles han aumentado el uso de pagos digitales, y seguirán haciéndolo en los próximos meses, aumentando el uso de aplicaciones de pago Online en 8 puntos porcentuales, según los expertos.

Así pues, con esta aplicación, se contribuye a la modernización de la sociedad y también, en parte, al minimizar el número de transacciones en moneda o billetes, se puede conseguir bajar en buena medida el riesgo de contagio de enfermedades que pueden producirse por el intercambio de efectivo.

# 4. Descripción y Objetivos

## Interfaz

La aplicación debe tener una interfaz intuitiva y lo más simple posible para que su utilización sea sencilla, y con los datos suficientes para tener controlado los gastos.

## Inicio

Se debe presentar una primera pantalla, en la que deberíamos introducir nuestro nombre, para que la aplicación pueda utilizar éste para añadir al usuario como propietario del “Wallet”, y también como miembro del mismo, para poder participar en las diferentes transacciones.

## Presentación de los “Wallets”

Debemos tener una pantalla, donde se listarán todos los “Wallets” de los que somos propietarios, y que pulsando sobre ellos, podamos realizar diferentes acciones, como editar, borrar, o entrar en él, para operar.

## Manejo de las transacciones

Al entrar en un “Wallet” debemos encontrarnos con una lista de las transacciones existentes, en las que de un vistazo podamos ver lo más importante, y también debemos encontrarnos con un pequeño resumen de las transacciones del “Wallet”. A su vez, podremos añadir, editar, borrar transacciones.

## Resolución de deudas

Ésta es la finalidad principal de la aplicación a partir de los datos introducidos, deberá ser muy intuitiva, con el mínimo de datos posibles, para no hacerla farragosa. En ella se incluirá, lo que cada miembro debería pagar a otro miembro, para poder saldar las deudas entre todos los miembros.

También deberá incluir un tipo de resumen de todas las operaciones totales por cada miembro.
Y deberá poder cancelarse la deuda entre los miembros de una forma sencilla.

El objetivo es que sea una aplicación que **Facilite saldar las deudas entre varias personas**, que realizan una actividad común de forma totalmente **automática**, y con la mínima intervención de los participantes.

**Gestionar el gasto de una actividad**, para control de ésta y poder llevar un control económico de dicha actividad.

**Analizar el coste final de dicha actividad**, y reparto de deuda entre los participantes.

**Calcular las deudas contraídas entre los participantes**, para que sean resueltas de una manera rápida, y con el menor intercambio de efectivo entre los mismos.

# 5. Contenidos

## Introducción

La aplicación manejará diferentes “Wallets”, estos “Wallets” pertenecerán a un **usuario**, de esta forma, en el futuro, cuando se pueda compartir la información, habrán acciones que sólo el **propietario** podrá ejercer. Para asignar correctamente este **propietario**, al iniciar la aplicación por primera vez, se nos pedirá un nombre de **usuario**.

Los “Wallets” están compuestos **miembros,** éstos son añadidos por el propietario al crear el “Wallet”, también se puede editar el “Wallet” para añadir o eliminar miembros posteriormente.

Las transacciones, a su vez, tendrán **participantes,** éstos se podrán elegir de los **miembros** que integran el “Wallet”, y serán aquellos que compartirán el gasto.

Cada transacción, implica un gasto, que abona el **pagador,** que es un **miembro** del “Wallet”, pero no tiene por qué ser un **participante** de dicha transacción obligatoriamente. Y de esta forma también, un **pagador**, puede ser el único **participante** de la transacción, otorgándose el gasto a sí mismo, pero eso si, siendo contabilizado en el total del “Wallet”.

## Icono

El icono expresa de forma clara y concisa la verdadera identidad de la aplicación, nos encontramos con una maleta de tamaño medio, que es atravesada por una aeronave, dando de un plumazo un significado de viaje, y a su vez, como parte importante, la maleta contiene un símbolo de Euro, lo que da referencia a la parte económica de dicha aplicación.

*Ilustración 1*

El color elegido ha sido una especie de magenta, que como es sabido tiene un significado de ayuda, que es lo que pretende esta aplicación, y trasmite sensación de entusiasmo, evidentemente, ese momento de salir de viaje o actividad.

## Interfaz

La aplicación tiene una interfaz intuitiva y lo más simple posible para que sea amena en su utilización. Se ha desechado información prescindible de las pantallas principales, y de esta forma se consigue una rápida comprensión de lo visualizado.

## Inicio

La primera vez que se ejecuta dicha aplicación, se crea la base de datos y se pide que introduzca un nombre de usuario para poder asignar los “Wallets” a dicho usuario.

## Presentación de los “Wallets”

La siguiente pantalla es el listado de Wallets. Aquí encontraremos un listado con los "Wallets", en los que podremos ver el nombre, la descripción, y a su derecha encontraremos el importe total de todas las transacciones del mismo. La primera vez que iniciamos la aplicación, se creará un “Wallet” de demostración.
Con una pulsación larga sobre un “Wallet” podremos editarlo, tendremos la oportunidad de editar tanto el nombre, la descripción, como los miembros. En la parte inferior derecha de la pantalla del listado de los “Wallets” hay un botón [+] y, al pulsarlo , se mostrará una nueva pantalla en la podremos añadir un nuevo “Wallet”.

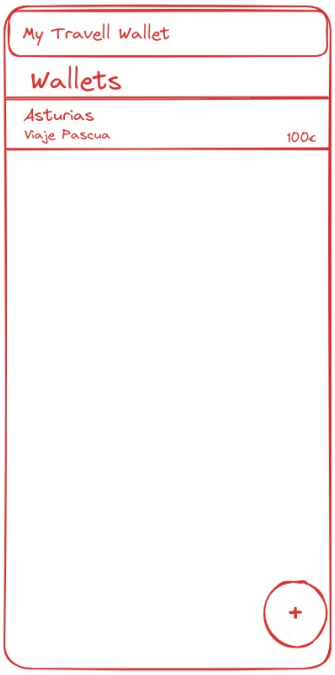



*Ilustración 2*

## Creación y edición de los “Wallet”

Al pulsar el botón con el signo [+] de la pantalla de listado de “Wallets”, se nos abrirá la pantalla de añadir nuevo “Wallet”, donde podremos añadir el nombre, una pequeña descripción, y con el botón de guardar se incluirá en la base de datos. En ese momento, nos aparecerá la lista de miembros, en un principio, como el “Wallet” es nuevo, aparecerá de forma predeterminada como miembro único el nombre del usuario de la aplicación. Podremos ingresar el nombre de los nuevos miembros, y pulsando sobre un icono con un signo más [+], que encontraremos a la derecha de la línea de ingreso, el nuevo miembro quedará reflejado en la lista. Podremos borrar un miembro mediante una pulsación

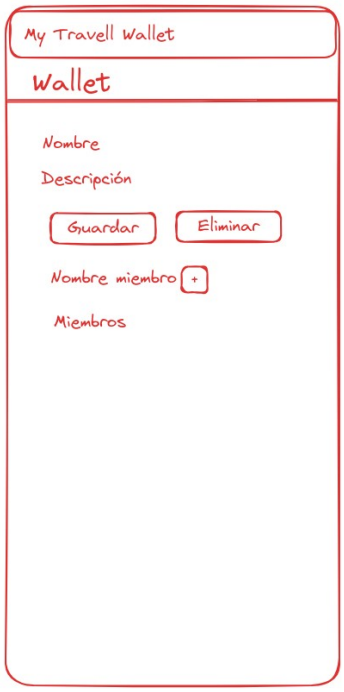



*Ilustración 3*

larga sobre el mismo, pero sólo se podrá borrar si no ha intervenido en ninguna transacción. También podremos editar el nombre con una pulsación corta, en la que nos aparecerá una pequeña ventana emergente, donde podremos ingresar el nuevo nombre, este nuevo nombre, quedará reflejado en toda la aplicación, así pues, si es miembro de otros “Wallets”, su nombre también variará en ellos.
Disponemos de un botón con el que podremos eliminar el “Wallet”, eso sí, siempre y cuando estén saldadas todas la deudas, de no ser así, se nos dará la opción de ir directamente a la pantalla de “Saldar Deudas” o bien cancelar la acción.
Pulsando el botón inferior derecho con un signo [X], volveremos a la pantalla de listado de "Wallets".
Si deseamos editar el “Wallet”, desde el listado de “Wallets” y mediante una pulsación larga, llegaremos a una pantalla, muy similar a la de añadir “Wallets”, donde del mismo modo, podremos modificar los datos pertinentes, añadir y eliminar miembros, y si lo deseamos, cambiar el nombre de los mismos o incluso el del “Wallet”.

## Manejo de las transacciones

Al pulsar sobre un “Wallet”, accederemos a la lista de transacciones. En la parte superior disponemos de cabeceras de columna, como descripción, categoría, importe, pagador y la fecha, si pulsamos dichas cabeceras, la lista se ordenará de forma ascendente y se volvemos a pulsar, lo hará de forma descendente.
En su parte inferior izquierda se indicará el gasto total de todas las transacciones, el nombre de todos los participantes y a su derecha se propondrá el nombre del próximo pagador, que será el que tenga más deuda con el resto de los participantes, abajo de éste, nos encontramos dos botones flotantes: [+] (añadir transacción) y [O] (saldar deudas).

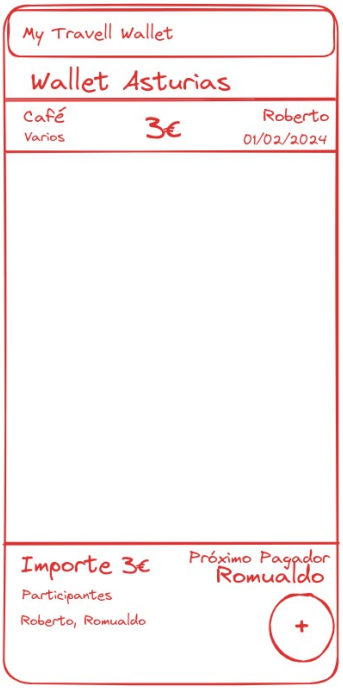



*Ilustración 4*

Si se pulsa sobre [+], añadir transacción, nos encontraremos diferentes apartados para ingresar, desde la descripción, importe, pagador, categoría y fecha de la transacción.

Como “pagador”, podremos elegir quién realiza el pago de entre los miembros agregados al crear el “Wallet”, de forma predeterminada, aparecerá seleccionado el participante que haya realizado menos pagos, al pulsar el espacio para ingresar el nombre, se desplegará una pequeña ventana emergente con los nombres de los miembros del “Wallet”, y a la derecha de cada miembro, aparecerá el importe total que ha pagado en dicho “Wallet”. Pulsando sobre uno de ellos, quedará seleccionado como pagador de la transacción.

En el apartado de categoría, escribiremos el nombre de la misma, y automáticamente nos aparecerán todas las categorías introducidas anteriormente que coincidan con las letras que estamos introduciendo. De forma predeterminada la categoría “Varios” será la que esté seleccionada.

La fecha será del día actual de forma predeterminada, y al pulsar sobre ella, mediante una ventana emergente, se podrá elegir de un calendario la fecha deseada. Al seleccionarla, quedará registrada en su apartado.

Justo debajo de la fecha, nos encontraremos una pequeña información de lo que deberían pagar cada uno de los participantes seleccionados en esa transacción. Según el número de los mismos, la división por participante irá cambiando, esta división es siempre a partes iguales entre todos los participantes.

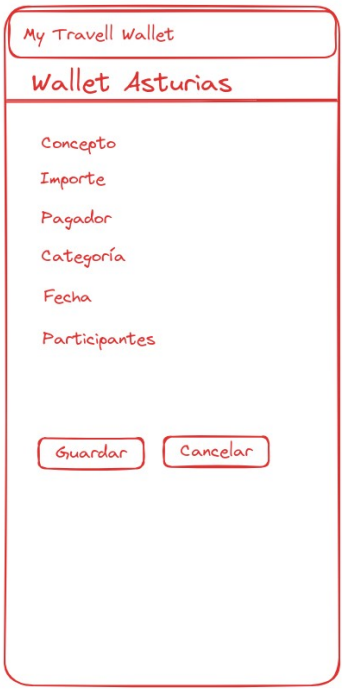



*Ilustración 5*

Los participantes, estarán de forma predeterminada, todos seleccionados de entre los miembros del “Wallet”, a su izquierda, mediante una casilla de verificación, se podrán incluir selectivamente. También cabe la opción de seleccionarse a uno mismo solo, de esta forma, podríamos llevar nuestra contabilidad individual siempre que incluyamos todas nuestras transacciones. Pero, hay que tener en cuenta, que estos gastos, sí quedarán reflejados en el total de “Wallet”.

En la parte inferior, tendremos dos botones, uno para guardar la transacción y otro para cancelar.

De vuelta en la lista de transacciones, si realizamos una pulsación corta sobre la transacción elegida, podremos editar dicha transacción, nos aparecerán todos los

apartados dispuestos exactamente igual que en la pantalla de agregar transacción. Una vez editada, pulsaremos sobre "Aceptar", volviendo a la pantalla del listado de transacciones.

Si hacemos una pulsación larga sobre una transacción, se procederá a borrarla, tras un aviso con un pequeño mensaje, en el que se podrá aceptar o cancelar esta acción.

## Resolución de deudas

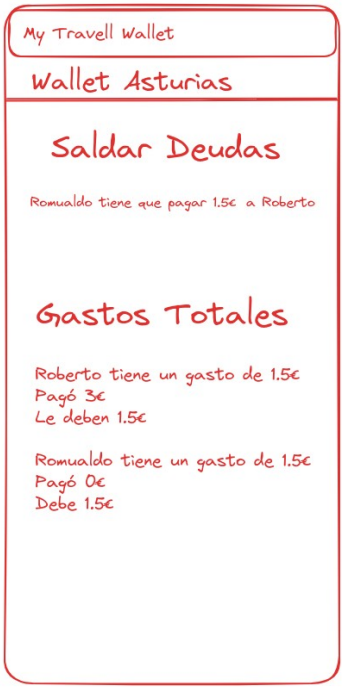



*Ilustración 6*

Si pulsamos sobre el botón [O], en la lista de transacciones, se mostrará una pantalla, en su parte inferior con el título de “Gastos Totales”, nos mostrará cada miembro del “Wallet”, junto a él, el importe de los gastos en los que participa, seguido del importe total que ha pagado, y el importe total que debe o le deben, siempre teniendo en cuenta su participación en las diferentes transacciones.

**En la zona superior, nos encontraremos con la parte más interesante de la aplicación, en la que se nos mostrará los diferentes miembros y el importe a saldar entre los mismos. Siempre se realizarán los diferentes cálculos para que, en el supuesto de haberlas, se salden las deudas haciendo el mínimo número posible de pagos entre los distintos miembros.**

# 6. Metodología y planificación

## Plan Operativo

La aplicación deberá resolver el problema de controlar el gasto, y saldar deudas, teniendo como meta, que sea totalmente funcional y de una manera muy intuitiva y fácil para todos los posibles usuarios, sean avanzados o noveles, se intentará al máximo, minimizar opciones y datos irrelevantes, así como tener unas pantallas limpias y de fácil comprensión.

## Análisis de Requisitos

Aplicación que facilitará el ajuste de cuentas entre varias personas de forma automática, donde se podrá gestionar y analizar el gasto de la actividad y calcular las deudas contraídas entre los miembros.

Se comenzará diseñando la base de datos, realizando el Diagrama de Entidad Relación ***Anexo 1***, posteriormente se empleará la clase SQLiteOpenHelper, para su uso.

Se continuará con el diseño y codificación de la “lista de transacciones”, primero la parte visual, creación del “layout”, con su “RecyclerView” donde se mostrará la lista de transacciones, una vez implementado su “Adapter”, seguidamente se creará el “Controller” para el acceso a la Base de Datos, mediante las clases de Android Studio, “SQLiteOpenHelper, SQLiteDatabase y Cursor, finalmente implementaremos las diferentes funciones en su “Activity”, tales como instanciar las vistas, configurar el “RecyclerView”, y los diferentes “Listeners” necesarios.

Seguiremos con la “lista de Wallets”, donde realizaremos las operaciones anteriores pero implementando los “Wallets”.

Por último y para finalizar, crearemos el “layout” de “Saldar Deudas”, y una vez implementado su “Activity”, instanciando sus vistas, y configurado su “RecyclerView” con sus diferentes “Listeners”, desarrollaremos la clase “DeudaUtility”, que

contendrá todas las operaciones necesarias para el correcto funcionamiento y resolución del apartado más interesante de esta aplicación, “Saldar Deudas”.

## Especificación Funcional

Se ha procedido a realizar una aplicación para Android, por lo tanto se ha escogido la línea 3 de las diferentes opciones proporcionadas para el proyecto fin de grado, esta elección se basa en la comodidad de disponer en el móvil del “Wallet” en todo momento, ya que actualmente, el móvil es una parte más de nosotros que nos acompaña a todas partes, y en todo momento.

La versión mínima de SDK elegida es la API 26, Oreo, versión de Android 8.0, es una versión lo suficiente “antigua” para funcionar en el 93,7% de los equipos como se puede observar en el ***Anexo 3,*** y lo suficiente “nueva” para poder implementar los últimos avances.

El IDE utilizado ha sido “Android Studio Jirafa” en un principio, y se actualizó a “Android Studio Iguana” posteriormente. Se ha elegido éste, por ser el oficial de Android.

La persistencia de datos se realiza, siguiendo lo aprendido en la asignatura “Base de Datos”, mediante el sistema de gestión de bases de datos relacional SQLite. Para las pruebas de sintaxis se ha utilizado la aplicación “DB Browser for SQLite” versión 3.12.1. Y para la implementación se han utilizado las clases “SQLiteOpenHelper, SQLiteDatabase y Cursor”.

## Implementación

Para el desarrollo, se ha utilizado control de versiones Git, y como repositorio remoto, se ha utilizado el repositiorio de GitLab, ***Anexo 3*** desde donde se puede acceder y comprobar dicho repositorio.

Se puede observar en el ***Anexo 4*** los diferentes “commits” realizados en el transcurso del desarrollo. Destacando la media de “commits” de 3.42 al día, con un máximo de 11 y un total de 147 “commits” en total en el desarrollo de la aplicación.

Se ha tratado de iniciar los commits con palabras cable para que de un vistazo tengamos claro cual ha sido la necesidad de ese commit, algunas palabras clave, son “Bug” para corrección de fallos, “Refactorizar” para refactorizado del código. “Implementación” para nuevas funciones.

De estos 147 “commits”, 14 son de “Refactorizado”, es decir, tomar trozos del código que funcionan correctamente y modificarlos para que de una forma mejorada realicen las mismas funciones, en ningún momento, la “Refactorización” nos solucionará problemas, pero hará que el código sea más legible.
Después de cada “Refactorización” se ha realizado una batería de pruebas, y la aplicación sigue funcionando correctamente.

También podemos encontrar 25 “commits” que tienen como inicio la palabra “Bug”, estos son errores en el código, que son descubiertos después de la batería de pruebas que se realiza cada nueva implementación. Su correcta subsanación es sin duda, uno de los mayores retos a los que nos podemos enfrentar, pero que mediante el “Debug” del IDE, y sus logs, podemos ir secuenciando el error, hasta llegar a su localización exacta.

**Planificación del proyecto**

El tiempo de desarrollo hasta la fecha es aproximadamente de 140h. En el ***Anexo 5*** podremos observar el desglose de forma concentrada.

Se ha seguido el “Modelo de Desarrollo por Etapas”, es muy similar al “Modelo de Prototipado”, en el que se van observando durante el desarrollo los diferentes estados, pero en este modelo durante el desarrollo del mismo, se van implementando nuevas especificaciones que no se tenían previstas, haciendo que el desarrollo sea vivo, y se vaya innovando.
Con este modelo de desarrollo podemos detectar también los fallos a tiempo para ser subsanados de inmediato, y a su vez tenemos una planificación de tiempos mucho más controlada.

## Pruebas

Durante el desarrollo de la aplicación, se ha realizado de forma periódica, una serie de pruebas para comprobar el correcto funcionamiento de la misma, y simular distintas acciones que puedan hacer los diferentes usuarios para intentar contemplar las distintas maneras de proceder con la aplicación. Se detallan las diferentes actuaciones realizadas en este apartado de pruebas.

1. Revisión de cada una de las “*Activity*” en busca de fallos, o código residual, posteriormente se han realizado diferentes baterías de pruebas específicas, mediante entradas de datos, y comprobada la correcta salida de los cálculos pertinentes.
2. Mediante el log debuger revisar posibles fallos ocultos, tales como fallos de memoria, o código repetitivo.
3. Se ha probado la instalación y funcionamiento en diversos equipos, tales como Tablet Xiaomi Mi Pad 5 de 11 pulgadas (Android 13, Miui 14), y en diferentes móviles, como Huawei P10 de 5,1 pulgadas (Rooteado, Android 13, LineageOS 20) , Huawei P30 Lite de 6,1 pulgadas (Android 12, EMUI 12), Samsung J510 de 5 pulgadas (Rooteado, Android 13, LineageOS 20), Xiaomi Redmi Note 10s de 6,43 pulgadas (Android 13, Miui 14), quedando descartado un Motorola G con Android 5.0, versión inferior a la mínima con al que se ha configurado esta aplicación, que es Android 8.0. Durante estas pruebas, aparte de comprobar el funcionamiento correcto, tanto de los métodos de introducción, como de los tiempos de respuesta. Se ha prestado mucha atención en la visualización de los diferentes objetos en las pantallas “*Layout*”, cada equipo tiene un tamaño de pantalla diferente, y se ha comprobado el correcto visualizado de todas las pantallas, y en la correcta implementación de las barras de desplazamiento en los listados “*ScrollView*”.
4. Creación de múltiples “Wallets”, cambio de nombre, y borrado de los mismos.
5. Añadidos múltiples miembros en los “Wallets”, cambio de nombre, y borrado de los mismos.
6. Añadidas múltiples transacciones con importes pequeños y grandes, con decimales muy variados.
7. Comprobada la “Resolución de deudas” de todas las pruebas efectuadas con ingreso, modificación de importes o modificación de participantes, con verificación de los resultados obtenidos mediante un cálculo manual de los mismos, es importante que todos los importes sean saldados correctamente.
8. Creación de transacciones de pagos a sí mismo. Variando importes, y mezclando estas transacciones con otras con diferentes miembros.

9. Creación de transacciones con pagos a otros, sin estar incluido el pagador, comprobación de los resultados mediante cálculo manual de los mismos.
10. Creación de transacciones con sólo alguno de los miembros y variando los participantes múltiples veces, para comprobar la respuesta de la aplicación ante tanta modificación.
11. Intento de eliminación de “Wallets” con los importes “no saldados”. Es una de las condiciones para poder eliminar un “Wallet”, primero se han de saldar todas las cuentas.
12. Borrado de miembros con transacciones realizadas, esto nunca será posible. Modificación de transacciones de dicho usuario, para comprobar si la respuesta es la correcta en todo momento.
13. Batería de pruebas también realizada por dos usuarios avanzados, y dos usuarios noveles que han verificado los resultados, y realizado un “Feedback” posteriormente.

## Mantenimiento

Tras recibir “Feedback” por parte de los usuarios noveles que han realizado las pruebas de funcionamiento, y los usuarios avanzados que han realizado una batería de pruebas con las mismas características, se han mejorado algunas etiquetas para mejor la comprensión, se han cambiado algunos textos de la ayuda, y solucionado varios pequeños bugs encontrados por los mismos.

# 7. Desarrollo

## Hardware

El equipo informático utilizado para el desarrollo, se centra en un portátil, de reciente adquisición expresamente para poder codificar de forma holgada en Android Studio, se trata de un  Pc Notebook HP EliteBook 250 G9 de 15,6 pulgadas, con un procesador Intel Core I7 1255U 12th, con 16 GB DDR4-3200 MHz de Ram. 512 GB PCIe NVMe SSD.
Se ha utilizado como pantalla secundaria la del portátil, y como pantalla principal una Acer X193W de 19 pulgadas.
Dispositivos de entrada, teclado Logitech K120 usb, y un ratón inalámbrico Vertical Zelotes F-36.

## Software

Como Sistema Operativo, se ha trabajado sobre Linux Mint 21.3 “Virginia” en su edición Cinnamon, esta versión está basada en Ubuntu 22.04.
El software IDE utilizado es Android Studio Jirafa en el inicio, y luego actualizando a la versión más reciente, Android Studio Iguana | 2023.2.1 Patch 1. Cabe destacar la utilización de la fuente “Fira Code” y habilitado las “font ligadures” para un visualizado más intuitivo del código.
Para la edición y pruebas de la base de datos, se ha utilizado “DB Browser for SQLite Versión 3.12.1”
En la edición del icono, y sus diferentes modificaciones, ha servido a la perfección el editor de imágenes vectorial Inkscape 1.1.
Para el diseño de los bocetos, y pantallas de ayuda, se ha utilizado la web ExcaliDraw.

## Estructura

Se inició mediante un “New Project – No Activity”, como nombre “My Travel Wallet, el “Package name” elegido, es “me.spenades.mytravelwallet”, como “Language” Java, “Minimum SDK” “API 26 (“Oreo”; Android 8,0), de “Build configuration language” “Kotlin DSL (build.gradle.kts) que era el recomendado.

Se ha dividido la estructura en diferentes “Packages”. Primero nos encontramos con “manifests” que incluye de forma predeterminada el archivo “AndroidManifest.xml”, “java” donde encontramos “res” con “drawable”, “layout” donde están incluidos todos los archivos de diseño de *layouts* en xml, “mipmap”, “values” aquí podemos encontrar los *resource* en xml, donde están referenciados los *styles*, *strings* y *colors* de la aplicación, y el *package* principal, *“me.spenades.mytravelwallet” en él están incluidos:*

1. *Activities:* Aquí podemos encontrar todas las *Clases Activity principales,* a excepción del ”MainActivity” que se encuentra en el raíz, las *activities* de ayuda y las de *utilities,* éstas clases muestran cada pantalla que veremos en la aplicación y que estará asociada a una interfaz de usuario, de esta forma se podrá interactuar con las pantallas de la aplicación (Nolasco, 2019), en esta clase especial, definiremos una clase que herede de *AppCompatActivity,* definiremos su *layout,* y también lo definiremos en el *manifest*.
2. *Adapters*: Aquí nos encontraremos las diferentes *Clases Adapters* que transformarán una interfaz en otra, en unión con *Recycler* será su proveedor de datos, puesto que es *Adapter* quien se los proporciona.
3. *Ayuda:* Tenemos las diferentes *Clases* que muestran las pantallas de ayuda la primera vez que utilizamos la aplicación.
4. Controllers: Mediante *model,* se transmitirá la información para ser almacenada en la base de datos, o recuperará información de la misma.
5. Models: Nutre a la vista y al controlador de la información de forma estructurada (Zigurd et al. (2011))
6. SQLiteDB: Aquí encontramos la Clase con la que *SQLiteOpenHelper (Robledo, 2017),* nos permitirá crear la base de datos, actualizar la estructura de tablas y los datos iniciales.
7. Utilities: En este *package* se ha incluido las diferentes clases para las operaciones, como *DeudaUtility,* que contiene las diferentes funciones y métodos para realizar todos los cálculos de la pantalla “Saldar Deudas”. También se incluye, la clase del *RecyclerTouchListener,* que es el *listener* de los diferentes *Recycler, PopUpPagadorActivity* es la clase que maneja la ventana emergente desde la cual elegimos el miembro que realiza el pago, *y DatePickerFragment,* con la que generamos el diálogo para seleccionar la fecha en una ventana emergente.

## Código

1. **Icono**

   En la edición del icono, si bien la creación se realizó mediante la web de diseño de logotipos *Brand Crowd,* luego se editó el resultado, añadiendo el símbolo de Euro, y realizando diferentes retoques mediante el editor de imágenes vectorial Inkscape 1.1, dándole también, ese color cercano al magenta sobre fondo blanco.

2. **Inicio**

   El primer paso al iniciar la aplicación, es comprobar si existe la Base de Datos, en caso positivo continuamos, y en caso negativo se crean las tablas necesarias para la persistencia de datos. Esta operación la realizamos con la clase *SQLiteOpenHelper*, ésta nos ayudará a crear, modificar, y conectar con la base de datos SQLite, con ella crearemos pues, la base de datos y su estructura. Definimos su constructor y sobreescribimos el método abstracto *OnCreate y onUpgrade*, en el primero, implementaremos el código necesario para crear la base de datos inicialmente, utilizando el método incluido en la API *execSQL,* éste método ejecuta directamente la orden SQL, en ella incluiremos la instrucción *“CREATE TABLE IS NOT EXISTS”* para que sólo cree la tabla si no existe ***Anexo 7.***

   Mediante el *controller* creado para manejar la tabla de usuarios, abrimos la base de datos en modo sólo lectura ***Anexo 8*** mediante el método *getReadableDatabase,* y utilizamos la clase *Cursor,* “esta clase permite acceder en modo lectura/escritura a los resultados devueltos por una consulta a la base de datos” (Robledo (2017), p.326). Iteramos el resultado del *Cursor y lo* incorporamos a un *ArrayList*, el tipo de elemento será el especificado mediante la clase *Usuario* de *models* ***Anexo 9.*** *Con el ArrayList creado, tenemos la lista de usuarios. Comprobamos el tamaño de la lista, si es 0 estamos ante un nuevo usuario, es decir, una instalación nueva, si el tamaño es mayor de 0, ya tenemos al menos un usuario en la base de datos, así pues, continuaremos a la pantalla de “Wallets”, en el caso de una lista tamaño*

*0, se nos presentará la pantalla de bienvenida a la aplicación* ***Anexo 10*** *, éste layout* inicial, consta de de un *ConstrainLayout,* se ha elegido este *layout* para que en todo momento los diferentes objetos se encuentren en la posición elegida, sea cual sea el tamaño de la pantalla del dispositivo en cuestión. Ésta pantalla, sólo es presentada la primera vez que se inicia la aplicación. Una vez se confirma el nombre del usuario mediante el botón de aceptar, mediante un *Intent “estructura de datos que contiene una descripción abstracta de una acción a realizar. Normalmente, lo usaremos para lanzar actividades” (Moreno, 2021, p.87)*, se cambiará de *Activity*, enviando mediante *Intent.putExtra,* el nombre de usuario y su Id. El siguiente *Activity* será la lista de *“Wallets”.*

3. **Wallets**

   En esta pantalla, como en las principales, cuando entremos por primera vez, se mostrará una imagen de ayuda ***Anexo 11***, una vez visitado, se guarda en la tabla ayudas la visita, y ya no se mostrará más veces.

   En el *Activity* definimos dos *Controller, walletController* con el que manejaremos el listado de “Wallets” y *transaccionController* que lo necesitamos para poder mostrar el importe total en la lista de “Wallets”, *instanciaremos* las vistas, crearemos las diferentes listas necesarias mediante *ArrayList,* configuraremos el *RecyclerView*, para ello, tras crear la clase *Model Wallet* y la clase *WalletController,* llamaremos a la función *obtenerWallets()* ***Anexo 12,*** *ésta* mediante la clase *SQLiteDatabase*, recuperaremos el listado de “Wallets” que se encuentra en la base de datos, ésta tabla, dispone de tres campos*. Una vez asignada la “listaDeWallets” a* el *ArrayList* correspondiente, lo implementamos en el adaptador *walletsAdapters.* A este *Adapter* también implementaremos el *ArrayList* con los importes del “Wallet” para poder mostrarlos en el listado. En el *Adapter* y mediante la Clase Abstracta *LayoutInflater (Wallace (2014)),* esta Clase es un inflador de diseños, esto es básicamente cuando tomamos una definición XML y la convertimos en objeto, creando una instancia del mismo, así pues, *creamos la nueva vista con* su *layout,* el cual contiene como raíz un *ConstraintLayout*. Para el *listener* del *RecyclerView,* se utiliza la clase *RecyclerTouchListener* para el *manejador*

táctil, el cual nos devolverá la posición pulsada (Diwakar S (2015)), y de esta forma podremos recuperar la información del “Wallet” deseado desde walletController.

4. **Transacciones**

En este *layout* ***Anexo 13***, tenemos un *FrameLayout* el cuál anida tres *LinearLayouts,* de los cuales, tanto el que contiene los indices de las columnas, como el resumen, lo harán mediante un *ConstrainLayout* y, con un *ScrollView* y un *LinearLayout,* albergará el *RecyclerView* de las transacciones. En su *Activity* definiremos *transaccionController y miembroWalletController,* para poder gestionar las transacciones y sus participantes.

Como podemos observar en el ***Anexo 14***, en esta ocasión se utiliza una sintaxis literal SQL directamente mediante el método *rawQuery,* que es mucho más flexible (Stroud, 2016), y puesto que se guardan el *Id* y no el nombre de los usuarios, se utiliza *Join* para recuperar el nombre de la tabla *Usuario*. Algo parecido ocurre con cada participante de la transacción, desde *ParticipaTransaccionController* obtendremos los *Id’s* de los miembros de los “Wallets”, los cuales se encuentran en la tabla *Wallet_Usuario.*

Para la elección del miembro que deberá pagar la transacción, se ha utilizado un menú *emergente* “...son generados sobre la marcha, pero el matiz que los diferencia de los contextuales es que las acciones de este menú no tiene efecto directo sobre el elemento donde aparecen” (Moreno, 2021, p.106).

Si pulsamos sobre alguno de los índices de los que conforman las columnas de las transacciones, podremos ordenarlas tanto de forma ascendente como descendente, como podemos ver en el ***Anexo 15***, utilizamos el método *Collections.sort (Charatan, 2019) de java.util.Collections,* el cual nos permite mediante los métodos *compare ó compareTo* ordenar la lista de transacciones. Para realizar esta función, al pulsar el *Listener* de los índices, se llama al procedimiento *ordenar,* el cual mediante el condicional de selección *switch,* ordenará el índice requerido.

En la parte inferior, tenemos el resumen de las transacciones, en este *ConstrainLayout,* se presentará el importe total de todas las transacciones que obtendremos desde el procedimiento *resumenTransacciones,* de la clase

*deudaUtility,* devolverá una lista con las transacciones que una vez sumadas, mediante *setText* lo incorporaremos al *Layout.*

5. **Resolución de deudas**

Esta parte del código es la que ha requerido mayor tiempo estratégico, si bien, después de muchos intentos infructuosos, se consiguió desarrollar un código funcional, eso sí, gracias a una pequeña ayuda de la IA (Inteligencia Artificial), la cuál, desde un principio, se consideró no utilizarla en ningún momento, porque los retos son realmente lo que hace que un aprendizaje sea efectivo, pero la imposibilidad de un resultado factible, hizo que al final se recurriera a ella, aunque, para ella fue también un gran reto, que tampoco terminó de resolver, pero la verdad, dio el punto de inicio, a un código, que poco a poco se fue desarrollando, y llegando a lo que hoy es el código final.

Éste es el código del resultado de la IA, ***Anexo 16,*** el cual no funcionaba correctamente, puesto, que directamente iba desechando a los miembros una vez dejaban de pagar deuda, pero no tenía en cuenta si le debía dinero otro miembro, además de otros problemas variados que hacía que no tuviera el final esperado, pero, éste código, fue la semilla del actual, en la que primero separamos lo que se tiene que pagar y lo que se tiene que cobrar, luego ordenamos los pagos de mayor a menor, de esta forma tendremos dos diccionarios, “a pagar” y “a recibir”, así conseguimos que el que debe más dinero, vaya pagando las deudas más pequeñas, y se van eliminando miembros de la ecuación mediante “*.remove”*, ésto lo conseguimos mediante iteración, es importante que aquellos que tienen que pagar, se iteren sobre los que tienen que recibir, se van pagando las deudas mediante la resta del importe a pagar, y eliminación del importe recibido al final mediante “*.replace”, reemplazando la cantidad inicial por 0*, de esta forma se van reduciendo las deudas contraídas, en el ***Anexo 17*** podemos observar las operaciones mencionadas. Iteramos sobre todos los miembros, gastos y pagos hasta que los diccionarios de “a pagar” y “a recibir” queden a 0 obligatoriamente ***Anexo 18.***

# 8. Conclusiones y proyección a futuro

Por el momento, la aplicación dispone de varias limitaciones, la más importante es que es offline, los “Wallets” no son compartidos, y por tanto, sólo un miembro podrá realizar las tareas sobre los mismos.

En un futuro sería importante implementar la base de datos en la nube, para poder compartir los “Wallets” entre los distintos miembros, y que cualquiera pudiera introducir dichas transacciones.

También se queda pendiente, una visualización correcta de los importes en las transacciones, se muestran con un punto en los decimales, y debería mostrarse con una coma, esto implica hacer una conversión a la hora de mostrar y recuperar dichos datos, puesto que el manejo que realiza el código, es mediante el punto en los decimales. Esta implementación se deja para una próxima mejora.

En sí, el desarrollo de la aplicación ha tenido sus puntos más complicados en el cálculo de la resolución de deudas, persistencia de datos, y en el manejo de los layout, para que se adaptara a diferentes tamaños de pantalla.

También el manejo de decimales, ha tenido momentos críticos, es muy importante que los importes queden perfectamente ajustados.

En general, este proyecto, me ha ayudado a afianzar lo aprendido en las diferentes asignaturas del grado. Desde el C.R.U.D. de la asignatura de “Base de Datos”, la elección de equipo, sistema operativo y configuración red, de la asignatura de “Sistemas Informáticos”, la adecuación de la parte visual gracias a “Desarrollo de Interfaces”, muy importante en la parte de cálculos ha sido lo aprendido en “Programación”, “Entornos de Desarrollo” para los primeros pasos con el IDE, elección del modelo de desarrollo, batería de pruebas y documentación. Y por supuesto lo aprendido en la asignatura fundamental de esta línea de proyecto, “Programación Multimedia y Dispositivos Móviles”.

# 9. Bibliografía

Charatan, Q & Kans, A. (2019). *Java in Two Semesters*. Springer. (Original publicado en 2002)

Diwakar, S. (2015). Recycler View Item Click Listener. *https://sapandiwakar.in/recycler-view-item-click-handler/*

*Moreno, V. (2021). Creación de aplicaciones con Android. Ra-Ma Editorial.*

*Moreno, V. (2021). Creación de aplicaciones con Android. Ra-Ma Editorial.*

Nolasco, S. (2019). *Desarrollo de aplicaciones con Android.* RA-MA Editorial.

Robledo, D. (2017). *Desarrollo de aplicaciones para Android*. Ministerio de Educación y Formación Profesional de España.

Robledo, D. (2017). *Desarrollo de aplicaciones para Android*. *Desarrollo de aplicaciones para Android*. Ministerio de Educación y Formación Profesional de España.

Stroud, A. (2016). *Android Database Best Practices.* Addison-Wesley Professional.

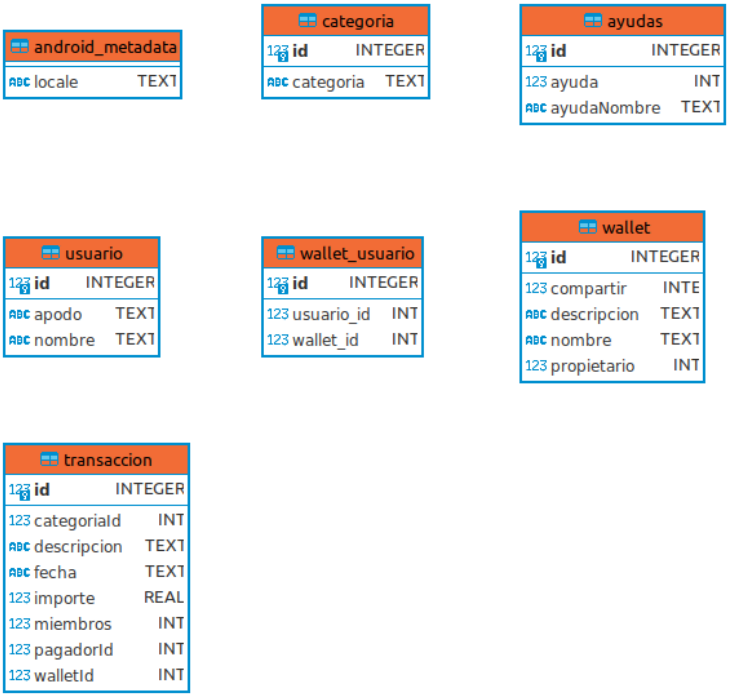
Wallace, J. (2014). *Pro Android UI.* Apress*,*

Zigurd, L., Dormin L., Blake G. & Nakamura M. (2011). *Programming Android.* O'Reilly.

# 10. Anexos

Anexo 1

*Diagrama ER*




Anexo 2:

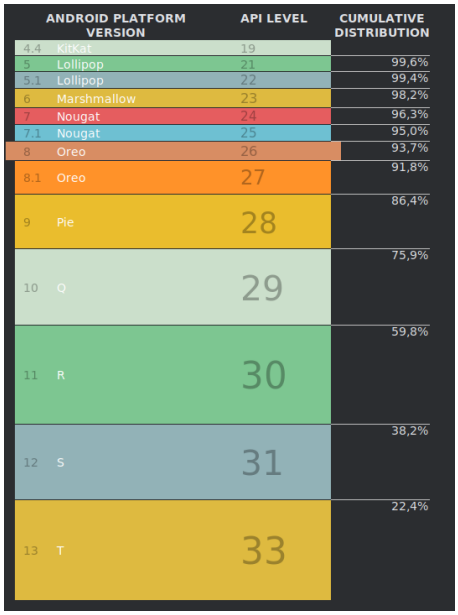
*Control de Versiones*

GitLab
User: proyectouniversaedam
Pass: 102023092024

(si es necesario pulsar “Update Email”)

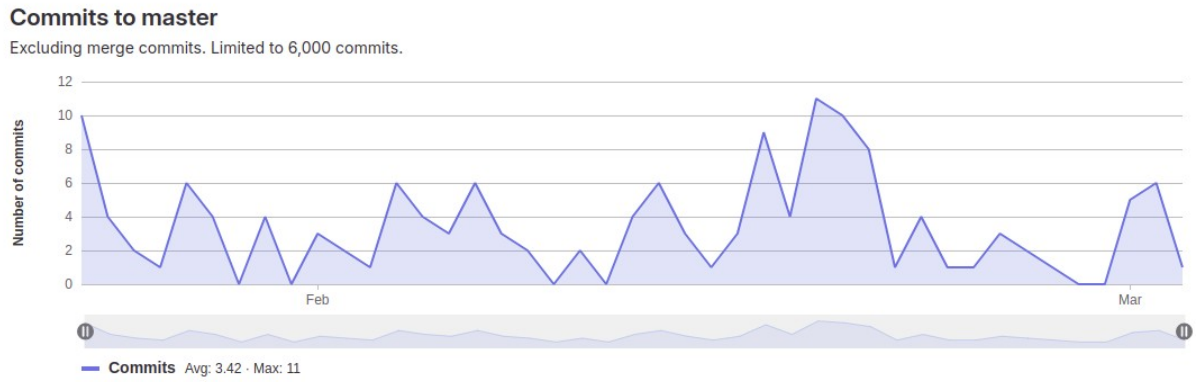
Anexo 3

*Nivel de Apis*



Anexo 4

*Contributor analytics*



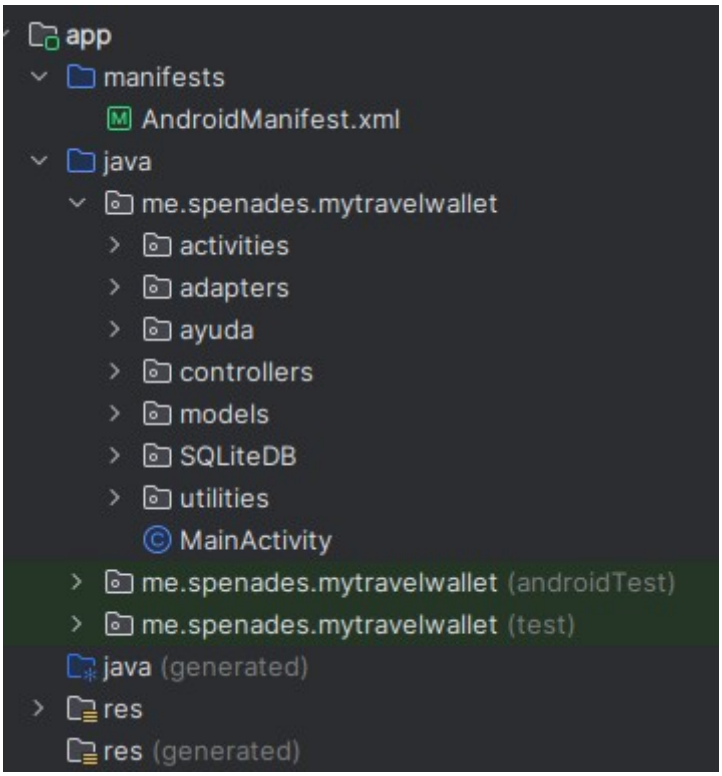

Anexo 5:

*Tiempo Desarrollo Aplicación*

| | Inicio | Fin | Horas |
|---|---|---|---|
| **Inicio del Proyecto** | 16/02/24 | 17/02/24 | 12h |
| **Diseño aplicación** | | 20/02/24 | 6h |
| **Desarrollo aplicación** | | 21/02/24 | 115h |
| | | **Total** | **133** |

Anexo 6

*Estructura app*




Anexo 7

*Código onCreate Base de Datos*

```
@Override
public void onCreate(SQLiteDatabase db) {

    db.execSQL(String.format("CREATE TABLE IF NOT EXISTS %s(id integer primary key autoincrement, nombre text unique, descripcion text, " +
            "propietario int, compartir inte)", NOMBRE_TABLA_WALLETS));

    db.execSQL(String.format("CREATE TABLE IF NOT EXISTS %s(id integer primary key autoincrement, descripcion text, importe real, pagadorId " +
            "int, miembros int, categoriaId int, fecha text, walletId int)", NOMBRE_TABLA_TRANSACCIONES));

    db.execSQL(String.format("CREATE TABLE IF NOT EXISTS %s(id integer primary key autoincrement, nombre text unique, apodo text)",
            NOMBRE_TABLA_USUARIOS));

    db.execSQL(String.format("CREATE TABLE IF NOT EXISTS %s(id integer primary key autoincrement, wallet_id int, usuario_id int)",
            NOMBRE_TABLA_WALLETS_USUARIOS));

    db.execSQL(String.format("CREATE TABLE IF NOT EXISTS %s(id integer primary key autoincrement, categoria text unique)",
            NOMBRE_TABLA_CATEGORIAS));

    db.execSQL(String.format("CREATE TABLE IF NOT EXISTS %s(id integer primary key autoincrement, ayuda int, ayudaNombre text)",
            NOMBRE_TABLA_AYUDAS));

}
```

Anexo 8

*Obtener usuarios*

```java
public ArrayList<Usuario> obtenerUsuarios() {
    ArrayList<Usuario> usuarios = new ArrayList<>();
    // readable porque no vamos a modificar, solamente leer
    SQLiteDatabase baseDeDatos = ayudanteBaseDeDatos.getReadableDatabase();

    // Los usuarios son de toda la app.
    String[] columnasAConsultar = {"nombre", "apodo", "id"};
    Cursor cursor = baseDeDatos.query(
            NOMBRE_TABLA,//from usuario
            columnasAConsultar,
             selection: null,
             selectionArgs: null,
             groupBy: null,
             having: null,
             orderBy: null
    );

    if (cursor == null) {
        /*
            Salimos aquí porque hubo un error, regresar
            lista vacía
         */
        return usuarios;

    }

    // Si no hay datos, igualmente regresamos la lista vacía
    if (! cursor.moveToFirst()) return usuarios;

    // En caso de que sí haya, iteramos y vamos agregando
    do {
        // El 0 es el número de la columna, como seleccionamos
        String nombreObtenidoDeBD = cursor.getString( columnIndex: 0);
        String apodoObtenidoDeBD = cursor.getString( columnIndex: 1);
        long usuarioIdObtenidoDeBD = cursor.getLong( columnIndex: 2);

        Usuario usuarioObtenidaDeBD = new Usuario(nombreObtenidoDeBD, apodoObtenidoDeBD, usuarioIdObtenidoDeBD);
        usuarios.add(usuarioObtenidaDeBD);

    } while (cursor.moveToNext());

    // Fin del ciclo. Cerramos cursor y regresamos la lista
    cursor.close();
    return usuarios;
```

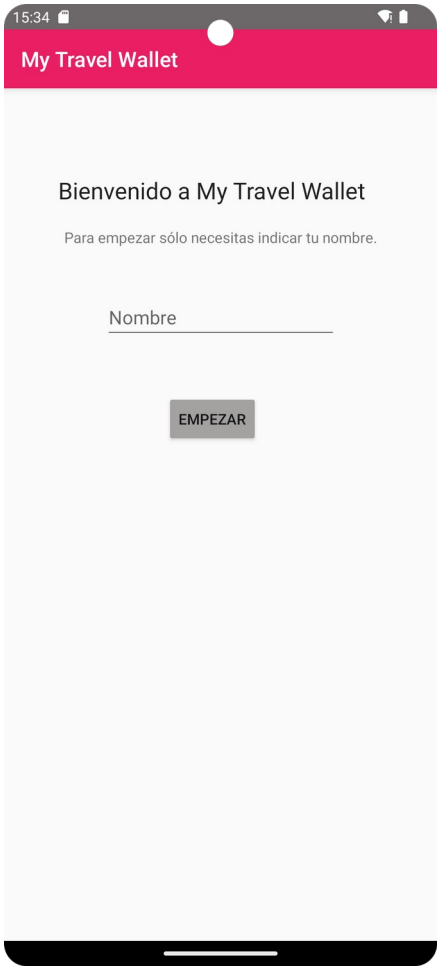
Anexo 9

*Usuario models*

```java
// Constructor para cuando instanciamos desde la BD
3 usages   Sergio Penadés
public Usuario(String nombre, String apodo, long id) {
    this.nombre = nombre;
    this.apodo = apodo;
    this.id = id;
}
```

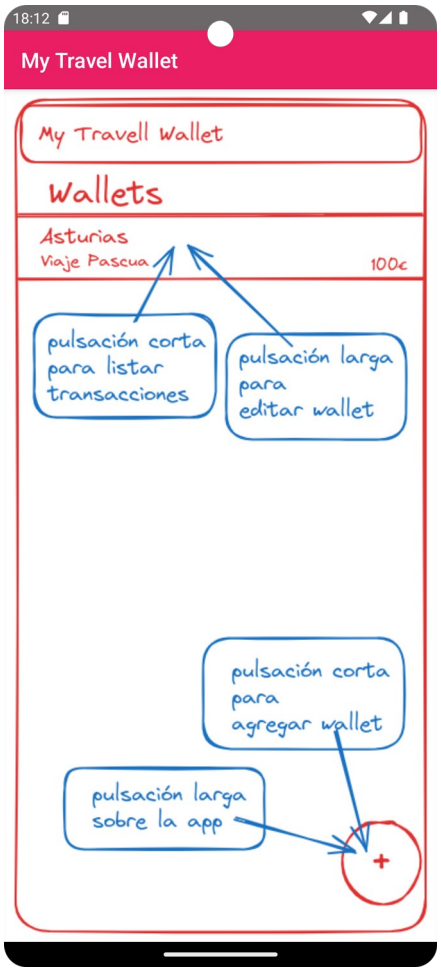
Anexo 10

*Pantalla de Bienvenida*



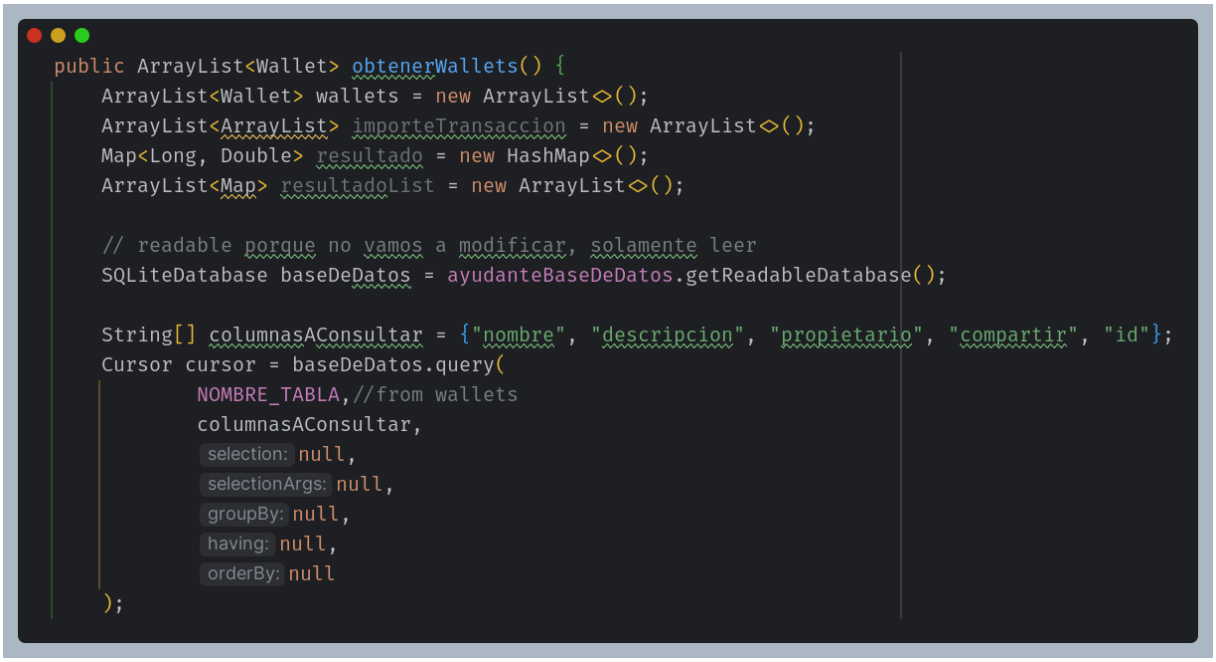

Anexo 11

*Ayuda lista wallets*



Anexo 12

*Obtener wallets*



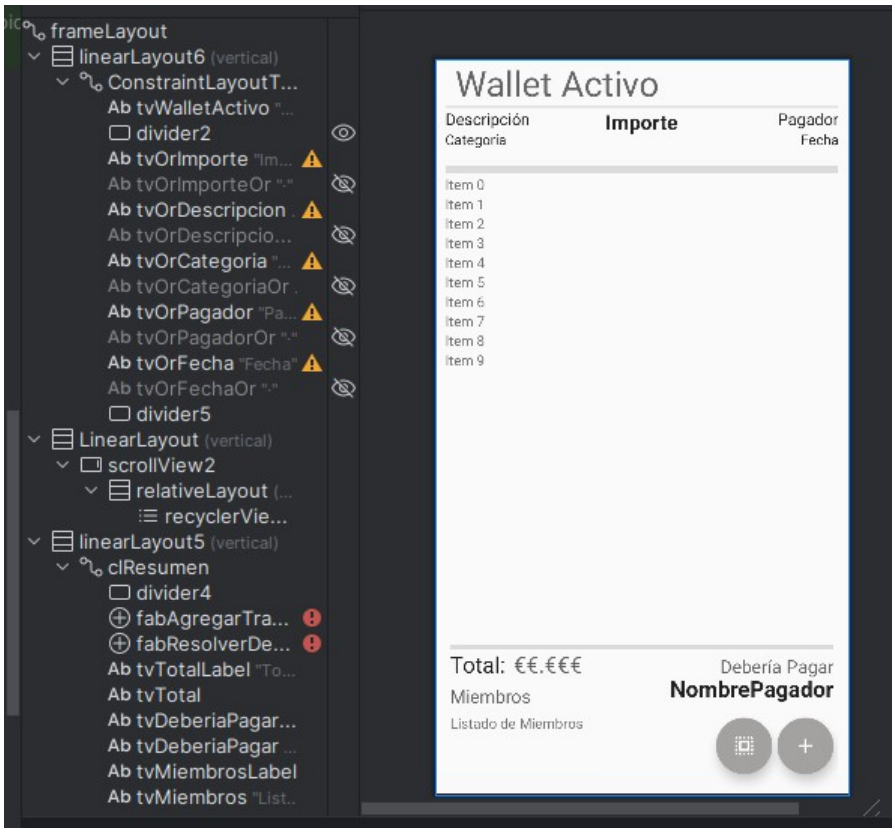
```
public ArrayList<Wallet> obtenerWallets() {
    ArrayList<Wallet> wallets = new ArrayList<>();
    ArrayList<ArrayList> importeTransaccion = new ArrayList<>();
    Map<Long, Double> resultado = new HashMap<>();
    ArrayList<Map> resultadoList = new ArrayList<>();

    // readable porque no vamos a modificar, solamente leer
    SQLiteDatabase baseDeDatos = ayudanteBaseDeDatos.getReadableDatabase();

    String[] columnasAConsultar = {"nombre", "descripcion", "propietario", "compartir", "id"};
    Cursor cursor = baseDeDatos.query(
            NOMBRE_TABLA,//from wallets
            columnasAConsultar,
            selection: null,
            selectionArgs: null,
            groupBy: null,
            having: null,
            orderBy: null
    );
```

Anexo 13

*Layout listado de transacciones*




Anexo 14

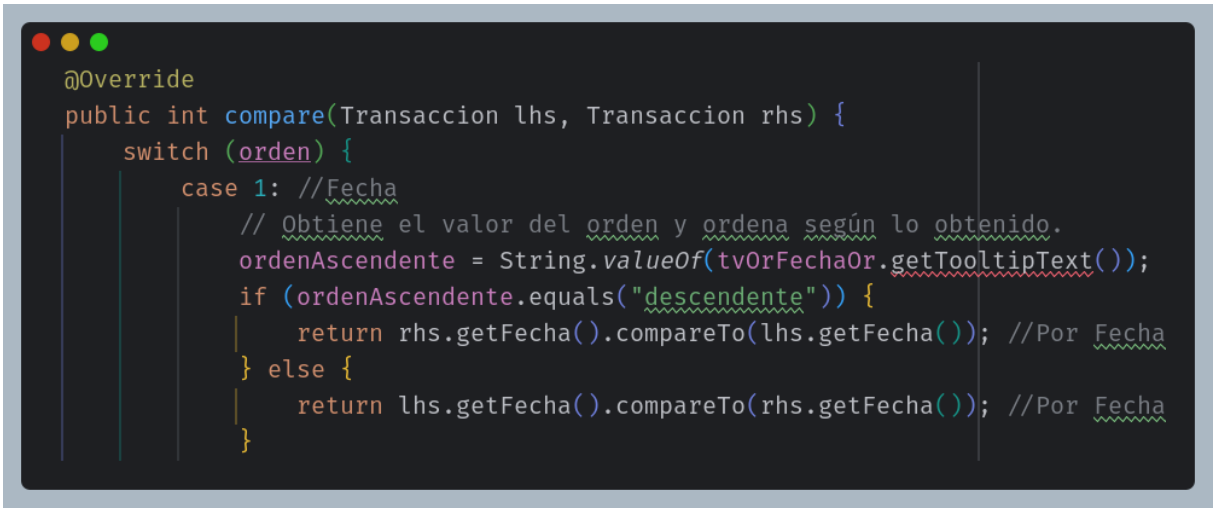
*Obtener transacciones*

```
String query = "SELECT descripcion,importe,pagadorId,miembros,categoriaId,fecha,walletId,TRANSACCION.id,nombre," +
        " CATEGORIA.categoria" +
        " FROM 'TRANSACCION' JOIN 'USUARIO' ON pagadorId = USUARIO.id JOIN 'CATEGORIA' ON categoriaId = CATEGORIA.id " +
        " WHERE walletId = " + WalletIdAConsultar;


Cursor cursor = baseDeDatos.rawQuery(query, selectionArgs: null);
```

Anexo 15

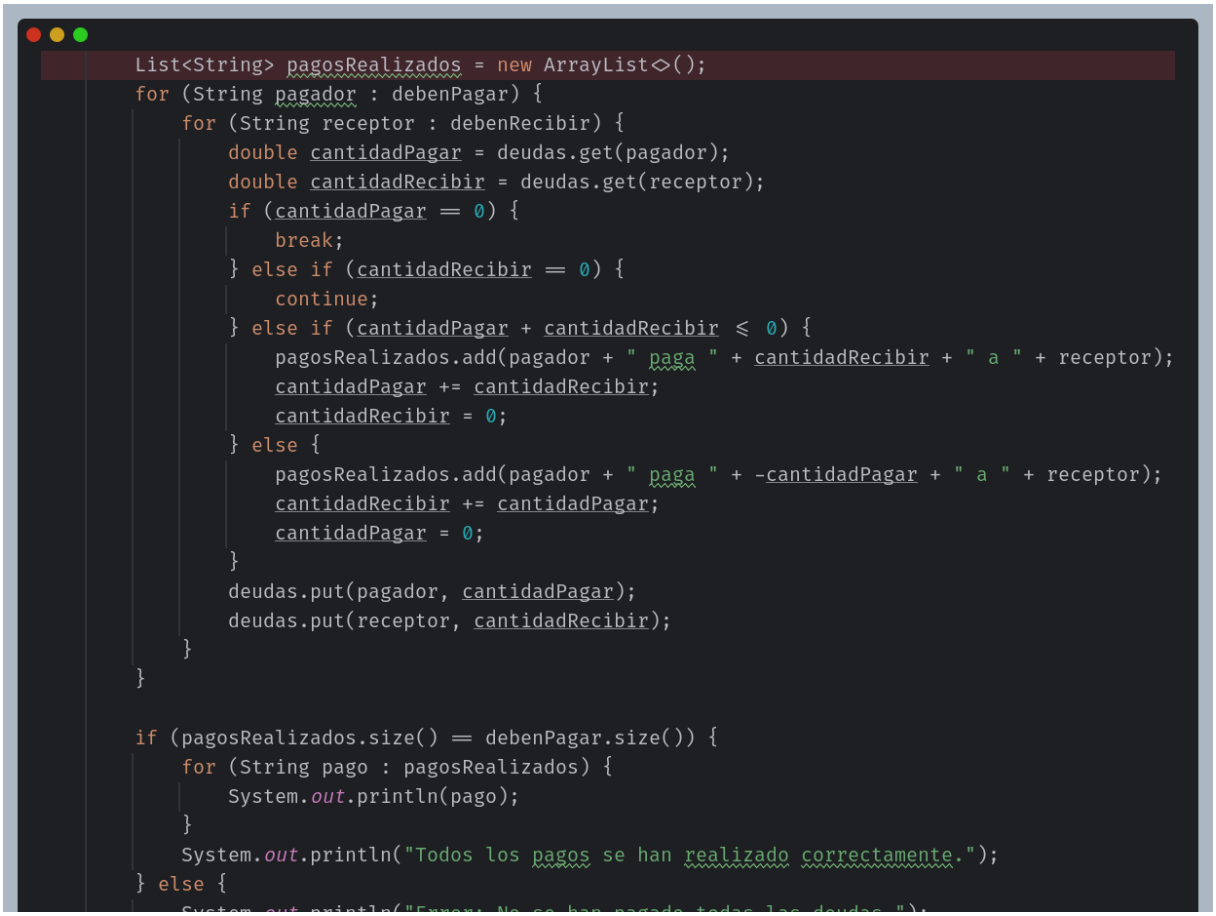
*Ordenar transacciones*



```
@Override
public int compare(Transaccion lhs, Transaccion rhs) {
    switch (orden) {
        case 1: //Fecha
            // Obtiene el valor del orden y ordena según lo obtenido.
            ordenAscendente = String.valueOf(tvOrFechaOr.getTooltipText());
            if (ordenAscendente.equals("descendente")) {
                return rhs.getFecha().compareTo(lhs.getFecha()); //Por Fecha
            } else {
                return lhs.getFecha().compareTo(rhs.getFecha()); //Por Fecha
            }
```

Anexo 16

*Código Saldar Deudas ChatGPT*

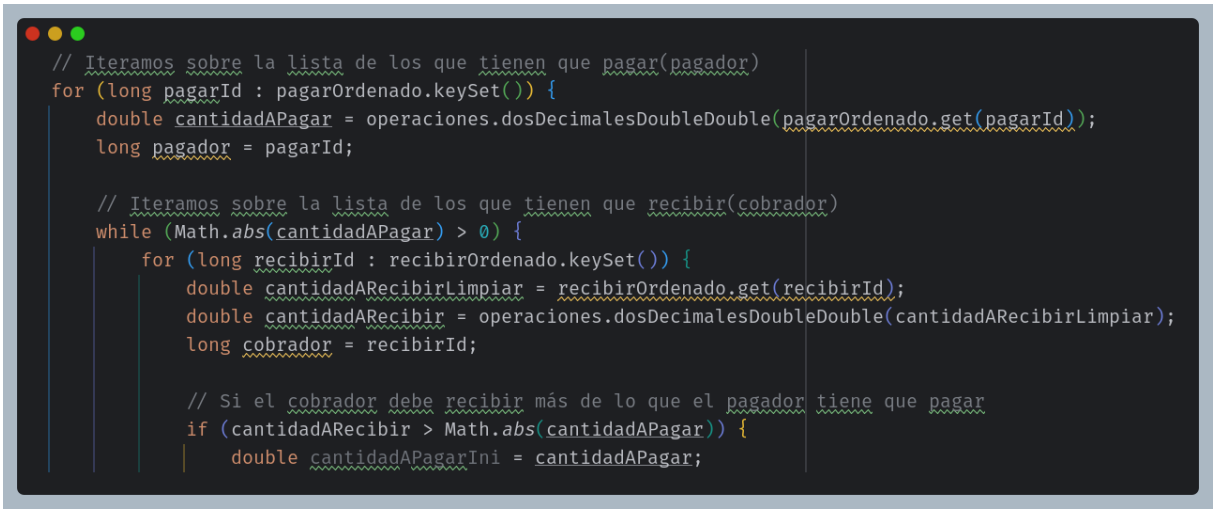


```
List<String> pagosRealizados = new ArrayList<>();
for (String pagador : debenPagar) {
    for (String receptor : debenRecibir) {
        double cantidadPagar = deudas.get(pagador);
        double cantidadRecibir = deudas.get(receptor);
        if (cantidadPagar == 0) {
            break;
        } else if (cantidadRecibir == 0) {
            continue;
        } else if (cantidadPagar + cantidadRecibir <= 0) {
            pagosRealizados.add(pagador + " paga " + cantidadRecibir + " a " + receptor);
            cantidadPagar += cantidadRecibir;
            cantidadRecibir = 0;
        } else {
            pagosRealizados.add(pagador + " paga " + -cantidadPagar + " a " + receptor);
            cantidadRecibir += cantidadPagar;
            cantidadPagar = 0;
        }
        deudas.put(pagador, cantidadPagar);
        deudas.put(receptor, cantidadRecibir);
    }
}

if (pagosRealizados.size() == debenPagar.size()) {
    for (String pago : pagosRealizados) {
        System.out.println(pago);
    }
    System.out.println("Todos los pagos se han realizado correctamente.");
} else {
```

Anexo 17

*Parte del Código de Saldar Deudas*



```
// Iteramos sobre la lista de los que tienen que pagar(pagador)
for (long pagarId : pagarOrdenado.keySet()) {
    double cantidadAPagar = operaciones.dosDecimalesDoubleDouble(pagarOrdenado.get(pagarId));
    long pagador = pagarId;

    // Iteramos sobre la lista de los que tienen que recibir(cobrador)
    while (Math.abs(cantidadAPagar) > 0) {
        for (long recibirId : recibirOrdenado.keySet()) {
            double cantidadARecibirLimpiar = recibirOrdenado.get(recibirId);
            double cantidadARecibir = operaciones.dosDecimalesDoubleDouble(cantidadARecibirLimpiar);
            long cobrador = recibirId;

            // Si el cobrador debe recibir más de lo que el pagador tiene que pagar
            if (cantidadARecibir > Math.abs(cantidadAPagar)) {
                double cantidadAPagarIni = cantidadAPagar;
```

Anexo 18

*Actualización de deudas*

```
                // Si el cobrador debe recibir menos o lo mismo de lo que el pagador
                // tiene que pagar
                if (Math.abs(cantidadAPagar) == 0) {
                    break;
                }
                if (Math.abs(cantidadAPagar) > 0) {
                    //double cantidadAPagarIni = cantidadAPagar;

                    // El pagador paga la cantidad que el receptor tiene que recibir
                    ArrayList<String> resoluciones = new ArrayList<>();
                    resoluciones.add(String.valueOf(pagador));
                    resoluciones.add(String.valueOf(usuarioIdNombbre.get(pagador)));
                    resoluciones.add(String.valueOf(cobrador));
                    resoluciones.add(String.valueOf(usuarioIdNombbre.get(cobrador)));
                    resoluciones.add(String.valueOf(Math.abs(cantidadARecibir)));
                    soluciones.add(resoluciones);

                    // Actualizamos la cantidad que el pagador tiene que pagar
                    cantidadAPagar += cantidadARecibir;

                    // El cobrador ya no tiene que recibir nada
                    recibirOrdenado.remove(cobrador, cantidadARecibir);

                    pagarOrdenado.replace(pagador, cantidadAPagar);
                    break;
                }
            }
```