

Marfol



**CICLO FORMATIVO DE GRADO SUPERIOR
DESARROLLO DE APLICACIONES MULTIPLATAFORMA**

Índice de contenidos

AGRADECIMIENTOS	3
RESUMEN	4
LICENCIA	5
1.INTRODUCCIÓN	6
2.NECESIDADES DEL SECTOR PRODUCTIVO	7
2.1 ANÁLISIS DE LA SITUACIÓN ACTUAL	7
2.2 NECESIDADES DEL CLIENTE Y OPORTUNIDADES DEL NEGOCIO	7
2.3 MARFOL – SIMPLIFICACIÓN DE LA DIVISIÓN DE CUENTA	7
3.DISEÑO DEL PROYECTO	8
3.1 FASES DEL PROYECTO	8
3.1.1 Análisis.....	8
3.1.2 Diseño.....	10
Colores.....	10
Base de datos	14
Arquitectura firestore.....	15
Arquitectura storage.....	19
3.1.3 Implementación.....	20
Inicio.....	20
Comprobar si el usuario está logueado	20
Index	21
Index -(Menú).....	25
Autenticación.....	27
Contraseña olvidada	28
Registro.....	29
Gestión de usuario	31
Home.....	32
Editar datos del usuario.....	34
Gestión de datos	36
Personas.....	37
Restaurantes.....	44
Platos	47
Participantes	51
Añadir participante	58
Añadir platos	59
Recordar platos.....	68
Añadir plato	71
Compartir lista	77
Detalle persona	85
Desglose	87
PROGRAMACIÓN DEL PROYECTO	95
JUSTIFICACIÓN DE VIABILIDAD	97
PREVENCIÓN DE RIESGOS.....	98

Agradecimientos

Kayler y Javier

Presentamos a **Luis Daniel Casado Guimaraes (DAW)**, un talentoso artista digital que ha dejado su huella en nuestro proyecto de manera extraordinaria.

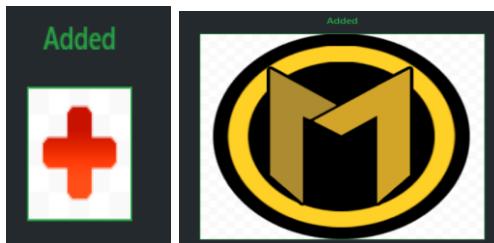
Daniel ha sido un pilar fundamental en la creación, mejora de imágenes y animaciones iniciales para nuestra plataforma.

Su talento como artista ha logrado mejorar y crear imágenes originales para Marfol.

Mejoras



Nuevas



Daniel ha logrado juntar los elementos gráficos que identifican Marfol y ha creado una animación que simplemente es impresionante.

El aporte creativo de Daniel ha sido invaluable para el éxito de nuestro proyecto, añadiendo un toque de magia a cada elemento visual.

Muchas gracias danieh



Resumen

El proyecto Marfol es desarrollado por configuradores, una organización no especificada que se dedica al diseño y desarrollo de aplicaciones móviles.

Marfol tiene como objetivo abordar las necesidades de simplificar y agilizar el proceso de dividir la cuenta después de una comida en un restaurante. Busca eliminar la confusión y complicaciones que surgen al compartir los gastos, brindando una herramienta intuitiva y eficiente para calcular los costos individuales de manera equitativa.

La posible demanda de Marfol son todos aquellos usuarios que frecuentan restaurantes y desean una solución práctica para dividir la cuenta de forma justa. Este servicio puede ser utilizado por individuos, grupos de amigos, familias u otros comensales que deseen una manera más fácil y precisa de realizar la división de gastos.

Marfol es una aplicación móvil que simplifica el proceso de dividir la cuenta después de una comida en un restaurante. La aplicación permite a los comensales ingresar fácilmente los platos y bebidas que han ordenado, asignando automáticamente los costos a cada individuo según los elementos seleccionados. Utiliza un algoritmo inteligente para calcular la división equitativa, teniendo en cuenta las preferencias de los usuarios. Además, ofrece características adicionales como la posibilidad de añadir elementos personalizados y compartir los resultados de la división de la cuenta a través de mensajes de texto o correo electrónico. Marfol también guarda un historial completo de comidas previas para facilitar el seguimiento y la documentación de gastos compartidos.

En conclusión, Marfol proporciona una solución innovadora y conveniente para simplificar la división de la cuenta en restaurantes, satisfaciendo las necesidades de los usuarios al eliminar la confusión y complicaciones asociadas con los gastos compartidos.

Licencia

Esta obra está bajo una licencia Reconocimiento-Compartir bajo la misma licencia 3.0 España de Creative Commons. Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-sa/3.0/es/> o envie una carta a Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

1. Introducción

Este documento se refiere a la realización del módulo de Proyecto del CFGS en Desarrollo de Aplicaciones Multiplataforma (DAM). El módulo de Proyecto complementa la formación establecida para los demás módulos profesionales que forman parte del título, enfocándose en el análisis del entorno, diseño del proyecto y organización de la implementación.

Marfol es una aplicación móvil diseñada para simplificar la división de la cuenta después de comer en un restaurante. Con su interfaz intuitiva, los comensales pueden ingresar fácilmente los participantes, platos, bebidas, y la aplicación se encarga de calcular los costos individuales de manera equitativa. Además, Marfol ofrece ajustes personalizados, como considerar situaciones especiales como compartir, y muestra resultados precisos para que cada persona pague su parte exacta.

En resumen, Marfol es la solución definitiva para dividir la cuenta en restaurantes. Con su funcionalidad eficiente, eliminando las complicaciones matemáticas y las discusiones interminables, Marfol simplifica el proceso y garantiza una experiencia sin complicaciones para todos los comensales. Ya no es necesario preocuparse por los gastos compartidos, ya que Marfol se encarga de todo de manera rápida, justa y sencilla.



2. Necesidades del sector productivo

2.1 Análisis de la situación actual

En este contexto, se analiza la situación actual relacionada con la división de la cuenta después de una comida en un restaurante. Se identifican las dificultades y complicaciones que suelen surgir al compartir los gastos, como las discusiones sobre quién consumió qué y el cálculo manual de los costos individuales. Este análisis destaca la necesidad de simplificar y agilizar este proceso.

2.2 Necesidades del cliente y oportunidades del negocio

Se identifican las necesidades de los clientes, que incluyen eliminar la confusión y complicaciones al dividir la cuenta, así como simplificar el proceso y asegurar una división equitativa. Estas necesidades crean una oportunidad de negocio para desarrollar una aplicación móvil como Marfol, que brinde una solución intuitiva y eficiente para calcular los costos individuales de manera justa y sin complicaciones.

2.3 Marfol – Simplificación de la división de cuenta

El proyecto Marfol propone una solución innovadora para simplificar la división de la cuenta después de comer en un restaurante. La aplicación móvil Marfol ofrece una interfaz amigable que permite a los comensales ingresar los platos y bebidas que han ordenado, asignando automáticamente los costos a cada individuo. Utilizando un algoritmo inteligente, Marfol calcula una división equitativa, considerando las preferencias de los usuarios y permitiendo ajustes personalizados. Además, Marfol ofrece funciones de personalización, compartición de resultados y registro de comidas anteriores, brindando una experiencia completa y satisfactoria para los usuarios. Con Marfol, la división de la cuenta se convierte en una tarea rápida y sin complicaciones, abordando las necesidades y demandas de los clientes en este sector.

3.Diseño del proyecto

3.1 Fases del proyecto

3.1.1 Análisis

En esta fase se establecerán los requisitos del proyecto, definiendo tanto las funcionalidades que debe incluir como las características y cualidades que deben cumplirse. En el caso de Marfol, se trata de una aplicación móvil para la división de la cuenta en restaurantes, por lo que se identificarán los requisitos específicos para su desarrollo. Estos requisitos se clasificarán en "Requisitos Funcionales", que describen las funcionalidades necesarias, y "Requisitos No Funcionales", que se refieren a las características y propiedades deseadas de la aplicación.

Requisitos funcionales

Marfol debe permitir a los usuarios registrarse y gestionar los nombres de los comensales. Los usuarios deben poder ingresar fácilmente los elementos de la orden y la aplicación debe realizar cálculos automáticos para dividir la cuenta de manera equitativa. Además, se deben proporcionar ajustes personalizados, una visualización clara de los resultados y la opción de compartirlos. Marfol también debe almacenar un registro de comidas anteriores para facilitar el volver a utilizar los mismos datos a futuro.

Requisitos no funcionales

Los requisitos no funcionales para Marfol incluyen la necesidad de una interfaz de usuario intuitiva y fácil de usar, un rendimiento rápido y preciso en el cálculo de la división de la cuenta, la confiabilidad y disponibilidad constante de la aplicación, la seguridad de los datos, la compatibilidad con diferentes dispositivos, la capacidad de escalar para manejar un crecimiento en usuarios, la facilidad de mantenimiento y actualización, y proporcionar una experiencia de usuario agradable y atractiva.

Requisitos Mínimos de Android:

Para garantizar un rendimiento óptimo y una experiencia fluida en la aplicación Marfol, se recomienda que los dispositivos Android cumplan con los siguientes requisitos mínimos:

Requisito	Especificación
Velocidad CPU	Procesador Quad Core de al menos 1.2 GHz o superior
RAM (GB)	Mínimo 2 GB de RAM recomendado
Cámara principal	Resolución mínima de la cámara principal de 3.0 MP (megapíxeles)
Almacenamiento Interno (GB)	Mínimo 1 GB de almacenamiento interno disponible
Almacenamiento Externo	Compatibilidad con tarjetas MicroSD para expandir la memoria (se recomienda capacidad de hasta 32 GB)
Localización	Dispositivo compatible con GPS (Sistema de Posicionamiento Global) para utilizar la geolocalización
Versión Android	Compatible con Android 7.X o versiones superiores (recomendado Android 7 o posterior)
Conexión	Requiere conexión Wi-Fi y 4G (servicio de datos) para acceder a los servicios utilizados por la aplicación

Requisitos Generales:

Además de los requisitos específicos de Android, se recomienda tener en cuenta los siguientes requisitos generales para garantizar un óptimo rendimiento y funcionamiento de la aplicación Marfol:

- Conectividad a Internet: Se requiere una conexión estable y confiable para acceder a todas las funciones de Marfol.
- Permisos de la aplicación: Se recomienda otorgar los permisos solicitados para el correcto funcionamiento de Marfol.
- Actualizaciones de la aplicación: Se recomienda mantener Marfol actualizado para tener acceso a las últimas mejoras y correcciones.
- Espacio de almacenamiento: Es necesario disponer de suficiente espacio en el dispositivo para almacenar datos de Marfol.

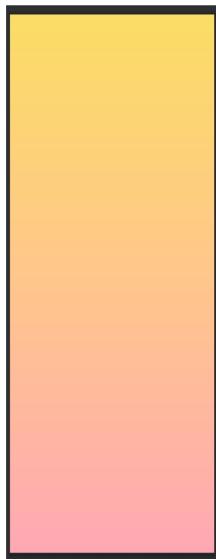
Estos requisitos generales son importantes para garantizar un uso óptimo de la aplicación Marfol y asegurar una experiencia positiva para los usuarios. Asegúrate de cumplir con estos requisitos y recomendaciones para obtener el máximo rendimiento y funcionalidad de la aplicación.

3.1.2 Diseño

En esta sección, presentamos el diseño visual de la aplicación Marfol, incluyendo los colores utilizados y la intención detrás de su elección.

Colores

Base:

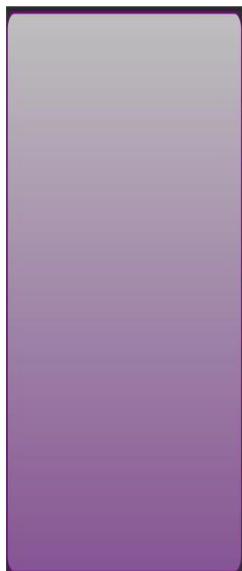


- #FFA8B4
- #FFC68C
- #FBDD65



Estos colores base se caracterizan por ser alegres, vivos y cálidos. Han sido seleccionados para transmitir una sensación de alegría y felicidad, ya que la aplicación Marfol se utiliza en momentos de compartir comidas y disfrutar de momentos especiales.

Platos:



- #7AE77EFF
- #bfbfbf



Los colores seleccionados para los platos se encuentran en una gama de tonos vibrantes y representan la diversidad y la variedad de opciones gastronómicas que ofrece la aplicación.

El inicio del color más claro se desvanece hacia un tono más intenso, lo que aporta un aspecto visual atractivo y moderno a los elementos relacionados con los platos.

Personas:



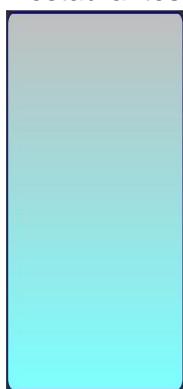
- #FBDD65
- #bfbfbf



Los colores utilizados para representar a las personas en la aplicación están basados en tonos cálidos y amigables. El color de inicio, un tono de amarillo dorado, simboliza la conexión y la interacción entre los comensales.

El desvanecimiento hacia un tono más claro en el final del color crea un efecto suave y armonioso.

Restaurantes:



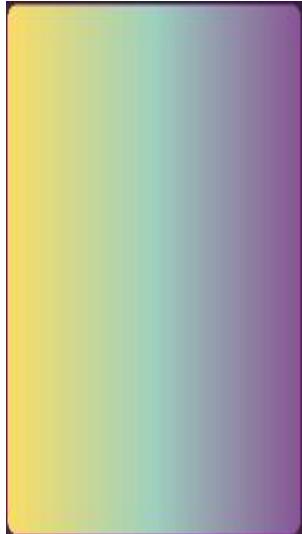
- #7EFFFF
- #bfbfbf



Los colores elegidos para los restaurantes se basan en tonos azules que evocan una sensación de frescura, tranquilidad y profesionalidad.

El degradado utilizado proporciona un aspecto moderno y agradable a la vista.

Historial:



- #FBDD65
- #9FD1C0
- #7AE77EFF



Los tres colores principales de la aplicación se usan para el historial, la fusión de colores representa todo el proceso de utilizar Marfol, desde añadir una persona hasta el restaurante con sus platos. Con esta paleta de colores, buscamos transmitir una sensación de armonía y conexión.



Los colores utilizados fueron seleccionados pensando en la naturaleza de la aplicación y su objetivo de proporcionar una experiencia agradable y positiva a los usuarios. Estos colores ayudan a crear una interfaz vibrante, alegre y fácil de utilizar, lo que resulta en una experiencia visualmente atractiva.

Además, mejoran la usabilidad de la aplicación al permitir que los usuarios identifiquen fácilmente los elementos en función de los colores elegidos.

Base de datos

En este proyecto, se ha decidido utilizar Firebase como la plataforma principal para el almacenamiento de datos y la autenticación de usuarios en la nube. Firebase ofrece una amplia gama de servicios, y en este caso, se han utilizado Authentication, Firestore y Storage para satisfacer las necesidades del proyecto.

Firestore es una base de datos NoSQL en tiempo real alojada en la nube que permite almacenar y sincronizar datos de manera eficiente. En el proyecto, Firestore se ha utilizado para gestionar la autenticación de usuarios y almacenar información relacionada con ellos.

- Firebase Authentication proporciona un sistema seguro y confiable para autenticar usuarios en la aplicación. A través de Authentication, se ha implementado un sistema de registro e inicio de sesión que permite a los usuarios crear una cuenta utilizando su dirección de correo electrónico y una contraseña segura.
- Firebase Firestore se utiliza para almacenar información relevante del usuario, como su nombre, dirección de correo electrónico y otros detalles personalizados. Cada usuario tiene su propio documento en Firestore, donde se guardan estos datos. Esto permite un acceso rápido y eficiente a la información del usuario y facilita su gestión y personalización.
- Firebase Storage es un servicio en la nube que permite almacenar y recuperar archivos, como imágenes o cualquier otro tipo de archivo, de manera segura y eficiente. En el proyecto, Storage se ha utilizado para gestionar el almacenamiento de imágenes relacionadas con los usuarios

Al registrarse o actualizar su perfil, los usuarios pueden cargar su imagen y esta se almacenará de forma segura en Storage. La URL de la imagen se guarda en Firestore, lo que permite recuperar y mostrar la imagen de perfil en la interfaz de usuario correspondiente, lo mismo se aplicará para platos y personas.

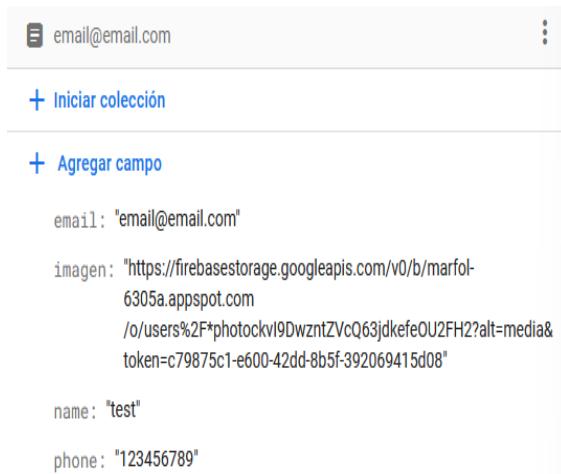
La elección de Firebase en este proyecto ha permitido un manejo eficiente de los datos y el almacenamiento de archivos. Gracias a las características y funcionalidades proporcionadas, los usuarios pueden registrarse y autenticarse de forma segura en la aplicación. Además, pueden almacenar y recuperar imágenes de perfil y otros archivos relevantes para su uso en la aplicación.

Arquitectura firestore

En este proyecto, se ha diseñado una estructura de base de datos utilizando Firestore para satisfacer las necesidades específicas. Se han creado cinco colecciones principales: "users", "personas", "restaurantes", "platos" e "historial".



Users: Almacena la información de los usuarios registrados en la aplicación.



- Email (obligatorio): Almacena la dirección de correo electrónico del usuario.
- Imagen (opcional): Permite al usuario almacenar una imagen de perfil.
- Name(opcional): Guarda el nombre del usuario.
- Phone(opcional): Almacena el número de teléfono del usuario (escala a futuro).

Personas: Se utiliza para almacenar los contactos de cada usuario. Cada documento

xE6NrJVMZ1pyRinjwGRd

+ Iniciar colección

+ Agregar campo

descripcion: "opcional"

imagen: "opcional"

nombre: "obligatorio"

usuarioId: "email@email.com"

- Descripción (opcional): Permite agregar una descripción adicional de la persona.
- Imagen (opcional): Permite almacenar una imagen de perfil para la persona.
- Nombre (obligatorio): Almacena el nombre de la persona.
- Usuariold(automático): Hace referencia al email del usuario al que pertenece esta persona.

Restaurantes: Almacena la información de los restaurantes asociados a cada usuario.

j7WZm1ynSQRkw5cii0B4

+ Iniciar colección

+ Agregar campo

nombreRestaurante: "test"

usuarioId: "email@email.com"

- NombreRestaurante (obligatorio): Almacena el nombre del restaurante.
- Usuariold (automático): Hace referencia al email del usuario al que pertenece este restaurante.

Platos: Se utiliza para almacenar información sobre los platos asociados a cada usuario y restaurante.

```

iErzNwVYoOjOVm8TcSFY
+ Iniciar colección
+ Agregar campo

descripción: "test"
imagen: "https://firebasestorage.googleapis.com/v0/b/marfol-6305a.appspot.com/o/platos%2FiErzNwVYoOjOVm8TcSFY.jpg?alt=media&token=7a0be18b-5cd4-4b90-a1e9-f03321493d3e"
nombre: "test"
precio: 2
restaurante: "test"
usuario: "email@email.com"
  
```

- Descripción (opcional): Permite agregar una descripción adicional del plato.
- Imagen (opcional): Permite almacenar una imagen para un plato.
- Nombre (obligatorio): Almacena el nombre del plato.
- Precio (obligatorio): Almacena el precio del plato.
- Restaurante (automático): Hace referencia al restaurante al que pertenece el plato.
- Usuario (automático): Hace referencia al email del usuario.

Los usuarios pueden almacenar varios platos con el mismo nombre mientras sean de un restaurante distinto.

Historial: Se utiliza registrar las instancias en las que el usuario ha utilizado los servicios de Marfol.

```

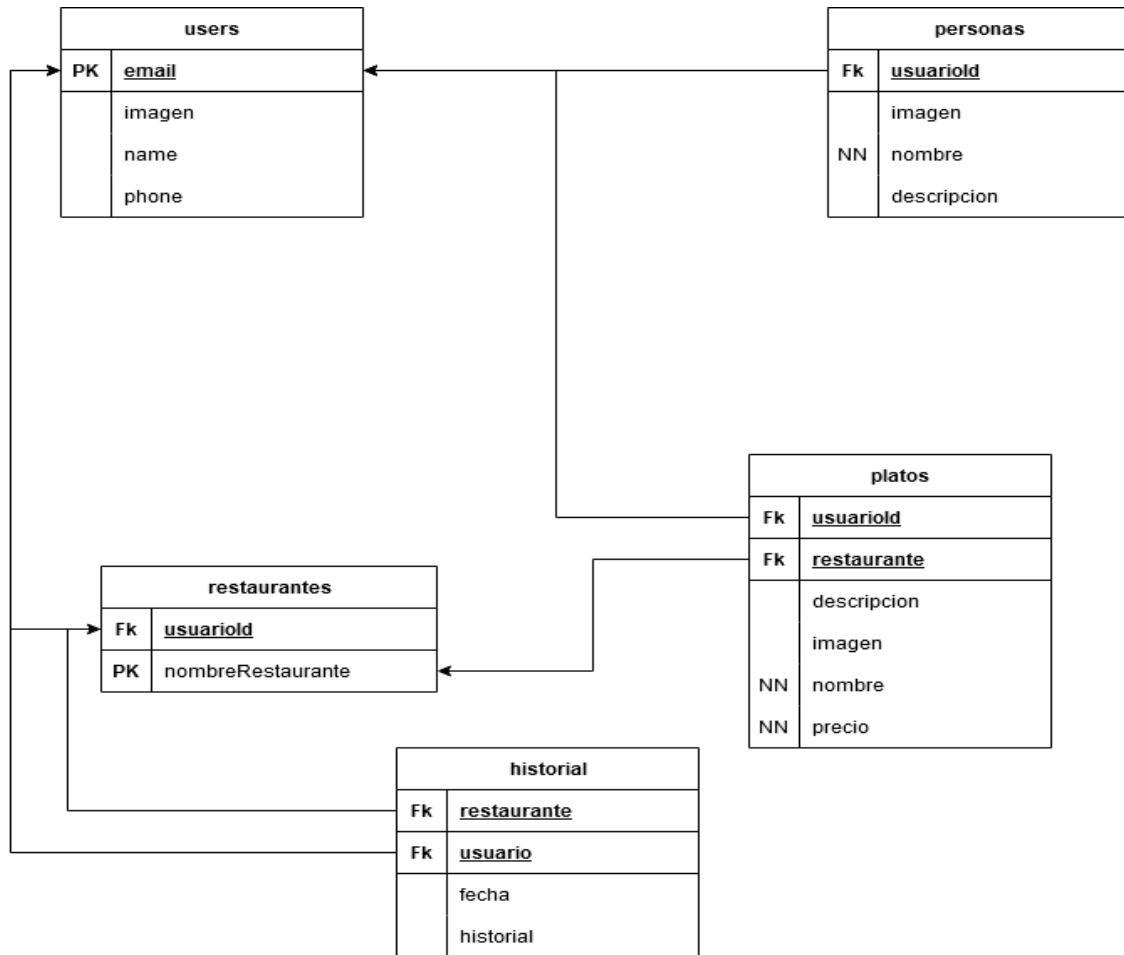
CLV22qcthQ5y8IPYxvPO
+ Iniciar colección
+ Agregar campo

fecha: "01/06/2023 08:35"
historial: "yo: 33.0 € lusia: 33.0 €"
restaurante: "barito"
usuario: "c@c.com"
  
```

- Fecha (automática): se guarda la hora y el día.
- Historial (automático): se guarda el nombre y el precio final de cada persona
- Restaurante (automático): se guarda el nombre del restaurante
- Usuario (automático): Hace referencia al email del usuario.

La base de datos ha sido diseñada de manera eficiente y coherente para satisfacer las necesidades del proyecto. Utilizamos el correo electrónico de cada usuario como identificador principal para filtrar la información de forma precisa. Los usuarios pueden almacenar varias personas con distintos nombres, y en caso de duplicados, se actualizan automáticamente. Lo mismo sucede con los platos y restaurantes, donde varios usuarios pueden tener registros con los mismos nombres debido a los identificadores únicos asignados. Esta estructura nos permite gestionar de manera adecuada y segura los datos, asegurando una experiencia fluida para los usuarios.

A pesar de que Firebase no es una base de datos relacional, hemos desarrollado un enfoque pseudorelacional utilizando el correo electrónico como elemento clave.



Arquitectura storage

Para el almacenamiento de imágenes se ha utilizado firebase storage.

<input type="checkbox"/>	Nombre	Tamaño	Tipo	Modificación más reciente
<input type="checkbox"/>	personas/	—	Carpeta	—
<input type="checkbox"/>	platos/	—	Carpeta	—
<input type="checkbox"/>	users/	—	Carpeta	—

Dentro del código se generan las carpetas para personas, platos y users.
Cada carpeta almacena sus respectivas imágenes

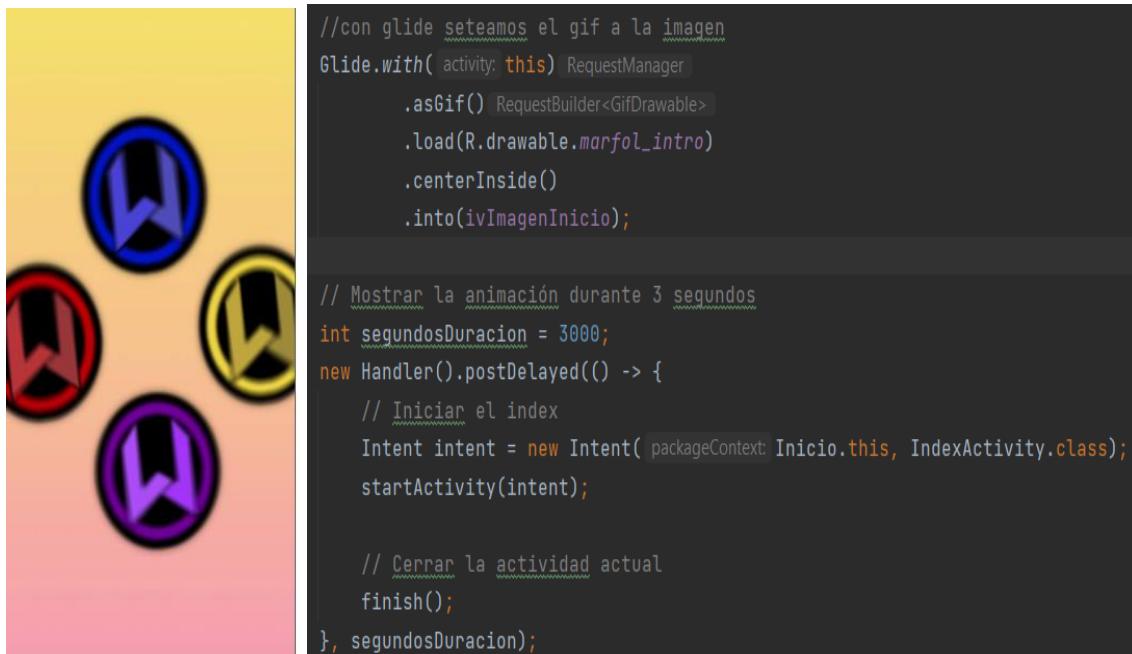
3.1.3 Implementación

Inicio

Marfol incluye una animación sencilla al abrir la aplicación, lo cual aporta un elemento estético y atractivo visualmente.

Cada vez que se inicia la app, se muestra una actividad con esta animación, que tiene una duración de 3 segundos.

Al finalizar la animación, se redirige automáticamente al index. Esta implementación brinda una experiencia más dinámica y agradable al usuario al interactuar con la aplicación.



Comprobar si el usuario está logueado

Antes de comenzar con el funcionamiento, es importante mencionar el método "comprobarLogueado" que se utiliza para verificar la existencia de una instancia de usuario. Esto permite cambiar la imagen del usuario.

Se emplea la librería "Glide" para cargar las imágenes. El método busca si la instancia del usuario está vacía y, en caso afirmativo, busca (en la colección "users") y actualiza la imagen del usuario. Este método recibe como parámetros el contexto (para su uso en varias actividades) y una imagen correspondiente a cada actividad en particular.

Se hacen las comprobaciones necesarias para evitar los fallos con la imagen.

```

public class MetodosGlobales {
    7 usages  ± Relyak +1 °
    public static boolean comprobarLogueado(Context context, ImageView ivImagen) {
        FirebaseAuth mAuth = FirebaseAuth.getInstance();
        FirebaseFirestore db = FirebaseFirestore.getInstance();
        if (mAuth.getCurrentUser() != null) {
            FirebaseUser currentUser = mAuth.getCurrentUser();
            DocumentReference userRef = db.collection("users").document(currentUser.getEmail());
            userRef.get().addOnCompleteListener(task -> {
                if (task.isSuccessful()) {
                    DocumentSnapshot document = task.getResult();
                    if (document.exists()) {
                        String imagen = document.getString("imagen");
                        if (imagen != null & !imagen.equalsIgnoreCase("") ) {
                            ivImagen.setPadding( left: 20, top: 20, right: 20, bottom: 20 );
                            ivImagen.setBackground(null);
                            try {
                                Glide.with(context).RequestManager
                                    .load(imagen).RequestBuilder<Drawable>
                                    .diskCacheStrategy(DiskCacheStrategy.ALL)
                                    .circleCrop() // Aplica el formato redondeado
                                    .into(ivImagen);
                            } catch (IllegalArgumentException e) {
                            }
                        } else {
                            try {
                                Glide.with(context).RequestManager
                                    .load(R.drawable.camera).RequestBuilder<Drawable>
                                    .diskCacheStrategy(DiskCacheStrategy.ALL)
                                    .circleCrop() // Aplica el formato redondeado
                                    .into(ivImagen);
                            } catch (IllegalArgumentException e) {
                            }
                        }
                    } else {
                        try {
                            Glide.with(context).RequestManager
                                .load(R.drawable.camera).RequestBuilder<Drawable>
                                .diskCacheStrategy(DiskCacheStrategy.ALL)
                                .into(ivImagen);
                        } catch (IllegalArgumentException e) {
                        }
                    }
                }
            });
        }
    }
}

```

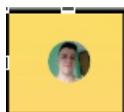
Este método resume muy bien la funcionalidad de firebase, comprueba si existe la instancia del usuario, apuntará a una colección con esa instancia y realizará una consulta.

Index

Actividad portada y raíz de la aplicación, desde la cual, se puede acceder a distintas funcionalidades a parte del inicio normal de la aplicación.



Imagen usuario

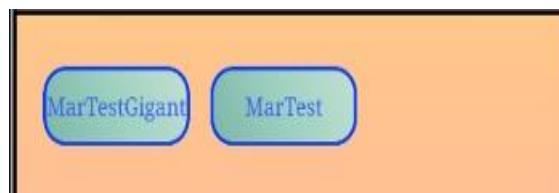


Da el acceso al conjunto de actividades relacionadas con la **autenticación**, registro, editar cuenta, etc.

Restaurantes

Restaurantes almacenados en la base de datos, el usuario tiene la oportunidad de recuperar de la base de datos los platos que habías añadidos anteriormente en ese restaurante

El usuario puede moverse entre los distintos restaurantes ya que se trata de una lista horizontal, al seleccionar un restaurante dará inicio a la aplicación, cargando todos los datos almacenados relacionados con el restaurante



¡PAGAR!



Botón principal que da inicio a la aplicación.

Google AdMob

Permite habilitar anuncios.

Solo mostrará el anuncio Test por el momento.



Index parte utilizando los siguientes métodos generales que, normalmente, comparten todas las actividades, son encargadas de asignar Ids a los elementos, asignar colores especiales a los textos.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_index);
    //Método que asigna IDs a los elementos
    asignarId();

    //Método que asigna efectos a los elementos (colores, etc)
    asignarEfectos();

    //si uno está logueado se comporta de una manera o otra
    comprobarLauncher();

    //Método que añade publicidad al index
    anadirAds();
```

asignarId

Asigna a las variables creadas los Ids de los elementos, incluyendo los elementos del popup, este método se repite en muchas clases.

```
public void asignarId() {
    //Asigna IDs a los elementos de la actividad
    btnApIndex = findViewById(R.id.btnApIndex);
    ivImagenLogin = findViewById(R.id.ivImagenLogin);
    Glide.with( activity: this ) RequestManager
        .load(R.drawable.nologinimg) RequestBuilder<Drawable>
        .diskCacheStrategy(DiskCacheStrategy.ALL)
        .into(ivImagenLogin);
    rvRestaurantesUsuario = findViewById(R.id.rvRestaurantesUsuario);
    tvTitleIndex = findViewById(R.id.tvTitleAnadirPlato);
    //Asigna IDs de los elementos del popup
    puVolverIndex = new Dialog( context: this );
    puVolverIndex.setContentView(R.layout.popup_confirmacion);
    btnCancelarIndex = puVolverIndex.findViewById(R.id.btnCancelarPopup);
    btnConfirmarIndex = puVolverIndex.findViewById(R.id.btnConfirmarPopup);
    tvMessage1Popup = puVolverIndex.findViewById(R.id.tvMessage1Popup);
    tvMessage2Popup = puVolverIndex.findViewById(R.id.tvMessage2Popup);
    tvTitlePopup = puVolverIndex.findViewById(R.id.tvTitlePopup);
    db = FirebaseFirestore.getInstance();
    mAuth = FirebaseAuth.getInstance();
    currentUser = mAuth.getCurrentUser();
}
```

asignarEfectos

Añade a los elementos de las actividades, efectos coloridos como degradados, cambios de color, etc.

```

public void asignarEfectos() {
    //Ajusta el tamaño de la imagen del login
    ivImagenLogin.setPadding( left: 20, top: 20, right: 20, bottom: 20);
    //Asigna el degradado de colores a los textos
    int[] colors = {getResources().getColor(R.color.redBorder),
                    getResources().getColor(R.color.redTitle)};
    float[] positions = {0f, 0.2f};
    LinearGradient gradient = new LinearGradient( x0: 0, y0: 0, x1: 40,
        tvTitleIndex.getTextSize(),
        colors,
        positions,
        Shader.TileMode.REPEAT);
    tvTitleIndex.getPaint().setShader(gradient);
    btnApIndex.getPaint().setShader(gradient);
    btnConfirmarIndex.getPaint().setShader(gradient);
    btnCancelarIndex.getPaint().setShader(gradient);
    // Asigna sombreado al texto
    float shadowRadius = 10f;
    float shadowDx = 0f;
    float shadowDy = 5f;
    int shadowColor = Color.BLACK;
    tvTitleIndex.getPaint().setShadowLayer(shadowRadius, shadowDx, shadowDy, shadowColor);
    btnApIndex.getPaint().setShadowLayer(shadowRadius, shadowDx, shadowDy, shadowColor);
}

```

comprobarLauncher

Método que comprueba la instancia del usuario si esta existe y cambia la funcionalidad de la imagen

```

private void comprobarLauncher() {
    if (MetodosGlobales.comprobarLogueado( context: IndexActivity.this, ivImagenLogin)) {
        botonImagenLogueado();
        mostrarAdapterRestaurantes();
    }
}

```

anadirAds

Inserta el anuncio tipo banner en la actividad.

```

1 usage  ▲ Javier Calderón
public void anadirAds(){
    MobileAds.initialize( context: this, initializationStatus -> {
        mAdView = findViewById(R.id.adViewIndex);
        AdRequest adRequest = new AdRequest.Builder().build();
        mAdView.loadAd(adRequest);
    });
}

```

cargarRestaurantes

Método encargado de recibir en una lista los restaurantes almacenados en la BD
 Se debe crear una lista contenedora y vamos añadiendo restaurante a restaurante a la lista.

Finalmente, insertamos en el adapter la lista de restaurantes para mostrarla en pantalla.

```

private void cargarRestaurantesBd() {
    if (currentUser != null) { // Verifica si hay un usuario actualmente logueado
        restaurantesBd = new ArrayList<>(); // Crea una nueva lista para almacenar los
        email = currentUser.getEmail(); // Obtiene el correo electrónico del usuario

        // Configura la referencia a la colección "restaurantes" en la base de datos
        restaurantesRef = db.collection(collectionPath: "restaurantes");

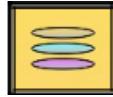
        // Realiza una consulta a la colección "restaurantes" donde el campo "userId"
        consulta = restaurantesRef.whereEqualTo(field: "userId", email);

        // Ejecuta la consulta y agrega un listener para recibir el resultado
        consulta.get().addOnCompleteListener(task -> {
            if (task.isSuccessful()) { // Verifica si la consulta se completó con éxito
                for (DocumentSnapshot document : task.getResult()) {
                    // Recorre los documentos resultantes de la consulta
                    nombreRestaurante = document.getString(field: "nombreRestaurante");
                    restaurantes = new Restaurantes(nombreRestaurante, userId: "");
                    restaurantesBd.add(restaurantes); // Agrega el objeto a la lista
                }
                restaurantesAdapter.setResultsRestaurantes(restaurantesBd); // Actualiza el adaptador
            }
        });
    }
}

```

Index –(Menú)

Te permite acceder a un menú desplegable que ofrece distintas funcionalidades, si estás logueado se acceden a más elementos.



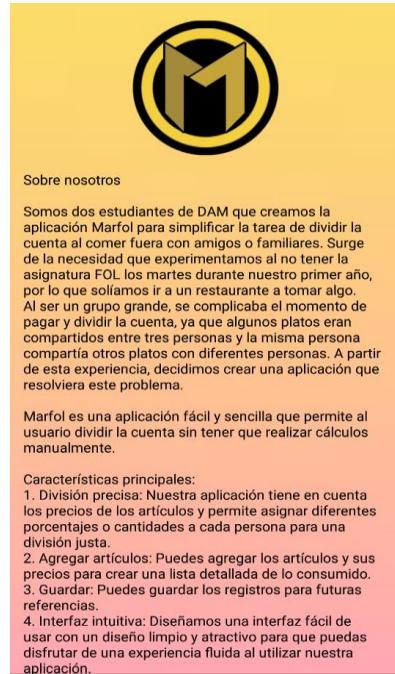
El menú contiene todos los colores utilizados en Marfol.



- Sobre nosotros (SIEMPRE)
- Editar (LOGUEADO)
- Historial (LOGUEADO)
- Logout (LOGUEADO)

Sobre nosotros

Inicia una actividad con un texto explicativo acerca de nosotros, los desarrolladores de Marfol, sobre nosotros aparecerá independientemente si el usuario está logueado.



Logout

Comprueba si el usuario está logueado, si lo está, habilita el botón Logout, su función es cerrar la sesión y reiniciar la actividad actual.

```
tvLogoutIndex.setOnClickListener(v -> {
    FirebaseAuth.getInstance()
        .signOut();
    intent = new Intent( packageName: this, IndexActivity.class);
    startActivity(intent);
    finish();
});
```

Editar e historial

Habilita los botones Editar – Historial únicamente si estás logueado, similar a Logout.

```
tvEditarDatosIndex.setOnClickListener(v2 -> {
    intent = new Intent( packageName: this, Seleccion.class);
    //startActivity(intent);
    seleccionCrud.launch(intent);
    popupWindow.dismiss();
});
tvHistorialIndex.setOnClickListener(v3->{
    intent = new Intent( packageName: this, HistorialBd.class);
    startActivity(intent);
    popupWindow.dismiss();
});
```

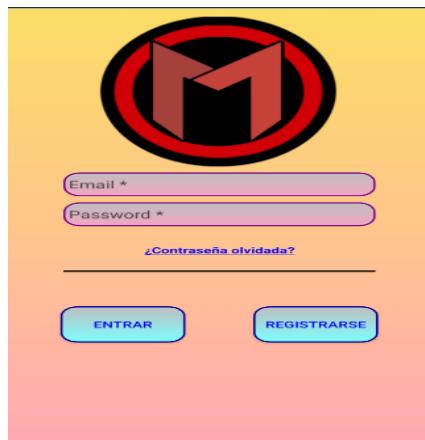
Autenticación

Marfol es una aplicación que ofrece la opción de crear una cuenta para almacenar datos, pero no es un requisito obligatorio. Los usuarios tienen la posibilidad de utilizar la aplicación sin necesidad de tener una cuenta registrada.

En Marfol, el usuario puede autenticarse en las actividades que contengan este ícono.



Al entrar en esta actividad te lleva al apartado de autenticación que contiene esta interfaz.



Esta actividad es una puerta de entrada a la aplicación para aquellos usuarios que ya tienen una cuenta creada. Su propósito principal es permitir un acceso rápido y sencillo. Se incluye la funcionalidad “contraseña olvidada” para aquellos usuarios que han olvidado su contraseña.

También se incluye la funcionalidad de registro para los nuevos usuarios.

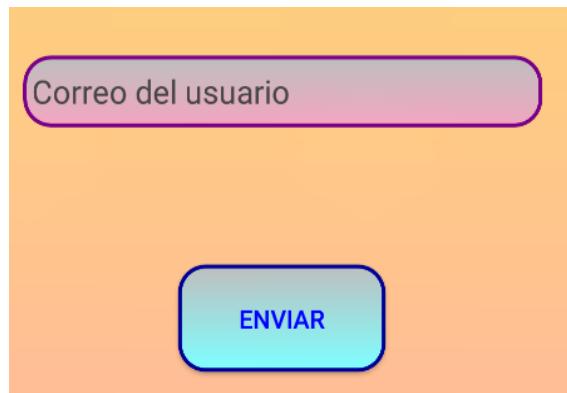
Para iniciar sesión, se realiza una consulta en Firebase utilizando los campos de correo electrónico y contraseña. Si el usuario existe y las credenciales son correctas, finaliza la actividad volviendo al index.

```
// Iniciar sesión con correo y contraseña
btnEntrarLogin.setOnClickListener(v -> {
    email = etEmailLogin.getText().toString();
    password = etPasswordLogin.getText().toString();
    if (!email.isEmpty() && !password.isEmpty()) {
        mAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener(it -> {
            if (it.isSuccessful()) {
                finish();
            } else {
                showAlert( title: "Error", message: "Se produjo un error en la autenticación del usuario");
            }
        });
    }
});
```

En caso contrario, se muestra un mensaje de error.



Contraseña olvidada



Para el proceso de recuperación de contraseña, se solicita al usuario que proporcione su correo electrónico. Si el correo electrónico proporcionado coincide con el de algún usuario registrado, se envía automáticamente un correo de recuperación desde la aplicación Marfol.

```
private void resetPassword() {
    //envía un correo al usuario de la caja de texto,(en caso de que el usuario esté registrado), y manda un reset de contraseña
    mAuth.setLanguageCode("es");//determina el idioma del correo
    mAuth.sendPasswordResetEmail(email).addOnCompleteListener(l->{
        if(l.isSuccessful()){
            Toast.makeText( context: this, text: "Se ha enviado un correo para restablecer la contraseña",Toast.LENGTH_SHORT).show();
        }else{
            Toast.makeText( context: this, text: "No se pudo enviar el email",Toast.LENGTH_SHORT).show();
        }
    });
}
```

Este correo contiene las instrucciones necesarias para que el usuario pueda restablecer su contraseña y acceder nuevamente a su cuenta.

Cambia la contraseña de Marfol Recibidos x

 noreply@marfol-6305a.firebaseio.com
para mí ▾

Hola:

Haz clic en este enlace para cambiar la contraseña de Marfol de tu cuenta: @gmail.com.

https://marfol-6305a.firebaseio.com/_auth/action?mode=resetPassword&oobCode=AEDFDXl_v4OifhwNMzopgnItaUD5C7eloYNmJNFgTe8AAAGiZ3ktGo&apiKey=AIzaSyC1jbQNq7WkFD4zGdVr1Y-RkaWluXSKE&lang=es

Si no has solicitado este cambio, ignora este correo electrónico.

Gracias,

El equipo de Marfol

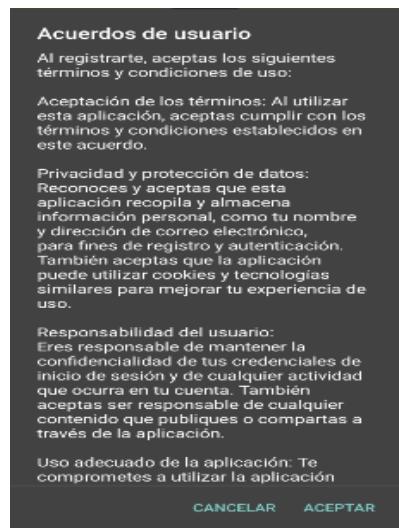
Registro

La actividad de registro en la aplicación tiene como objetivo permitir a los usuarios crear una nueva cuenta.



Al iniciar esta actividad, se presenta una interfaz de usuario intuitiva que solicita al usuario que proporcione su dirección de correo electrónico y una contraseña. Estos datos son esenciales para crear una cuenta personalizada y segura.

Los campos obligatorios en esta actividad son el email, contraseña (que tiene que cumplir unos requisitos mínimos de seguridad) y los acuerdos de usuario.



Al registrarte en la aplicación, el usuario acepta los términos y condiciones de uso que incluyen la aceptación de los términos establecidos, el manejo de privacidad y protección de datos, la responsabilidad del usuario, el uso adecuado de la aplicación y el reconocimiento de la propiedad intelectual.

Se han incorporado varias validaciones para evitar fallos.

```
// Validación de acuerdos
if (!cbAcuerdoUsuario.isChecked()) {
    // Mostrar mensaje de error
    showAlert("Debes aceptar los acuerdos de usuario");
    // Mostrar cuadro de diálogo con los acuerdos
    showAgreementDialog(cbAcuerdoUsuario);
    return;
}

if (email.isEmpty()) {
    // Validación de campo de correo electrónico vacío
    showAlert("El campo de correo electrónico está vacío");
    return;
}
if (!password1.equals(password2)) {
    // Validación de contraseñas distintas
    showAlert("Las contraseñas no coinciden");
    return;
}
if (password1.isEmpty()) {
    // Validación de campo de contraseña vacío
    showAlert("El campo de contraseña está vacío");
    return;
}
// Validación de contraseña con expresión regular
if (!password1.matches(REGEX)) {
    showAlert("La contraseña debe tener al menos 8 caracteres, un número y un carácter especial");
    return;
}
```

En caso que ocurra algún tipo de error, se mostrará un mensaje personalizado para cada caso.



Una vez que el usuario ha ingresado los campos obligatorios y es válido, se realiza el registro con el botón “REGISTRARSE”.

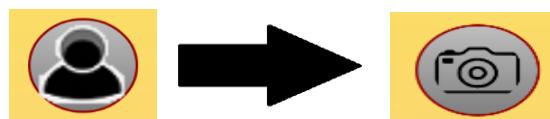
```
//Conexión a firebase, registro de usuarios con email y contraseña
FirebaseAuth.getInstance()
    .createUserWithEmailAndPassword(etRegistroEmailLogin.getText().toString(),
        etRegistroPasswordLogin.getText().toString()).addOnCompleteListener(it -> {
    if (it.isSuccessful()) {
        email = etRegistroEmailLogin.getText().toString();
        nombre = etNombreRegistro.getText().toString();
        telefono = etTelefonoRegistro.getText().toString();
        //mapeo de datos
        Map<String, Object> datosPersona = new HashMap<>();
        datosPersona.put("email", email);
        datosPersona.put("name", nombre);
        datosPersona.put("phone", telefono);
        datosPersona.put("imagen", "");
        db.collection( collectionPath: "users").document(email).set(datosPersona)
            .addOnSuccessListener(anadido -> {
                finish();
            }).addOnFailureListener(error -> {
                showAlert("Error al guardar los datos del usuario");
            });
    } else {
        showAlert("Ocurrió un error a la hora del registro");
    }
});
```

A su vez, se crea dentro de Firestore un documento con los datos del nuevo usuario.

Después del registro el usuario es redirigido al index.

Gestión de usuario

Una vez logueado, se modificará la funcionalidad de la imagen, cargando la imagen del usuario y si no la tiene cargará una por defecto.



Para cambiar la funcionalidad de la imagen primero se comprueba si el usuario está logueado.

```
private void comprobarLauncher() {
    if (MetodosGlobales.comprobarLogueado(context: IndexActivity.this, ivImagenLogin)) {
        botonImagenLogueado();
        mostrarAdapterRestaurantes();
    } else {
        Glide.with(activity: this) RequestManager
            .load(R.drawable.nologinimg) RequestBuilder<Drawable>
            .diskCacheStrategy(DiskCacheStrategy.ALL)
            .into(ivImagenLogin);
        botonImagenNoLogueado();
    }
}
```

Después cambia la funcionalidad con el método *botonImagenLogueado* que simplemente enviará al login o a la gestión del usuario. En el caso contrario enviará al usuario a la autenticación.

```
private void botonImagenLogueado() {
    //El usuario está logueado
    ivImagenLogin.setOnClickListener(view -> {
        intent = new Intent(packageContext: this, login.HomeActivity.class);
        rLauncherLogin.launch(intent);
    });
}

private void botonImagenNoLogueado() {
    // El usuario no está logueado
    ivImagenLogin.setOnClickListener(view -> {
        intent = new Intent(packageContext: this, login.AuthActivity.class);
        rLauncherLogin.launch(intent);
    });
}
```

Home

Lo primero que hará dentro de esta actividad es comprobar si el usuario está logueado, *comprobarLogueado*, y traerá los datos de firebase.



El usuario podrá cambiar la foto o sus datos con el botón editar que llevará a otra actividad.

En caso de entrar a esta actividad sin estar logueado (error) la actividad finalizará, evitando posibles fallos.

El método `cargarDatosEnHome` hará una consulta en la base de datos para cargar los campos nombre, teléfono y correo de cada usuario.

```
private void cargarDatosEnHomeSiLogueado(TextView email, TextView nombre, TextView telefono) {
    if (mAuth.getCurrentUser() != null) {
        // Obtener el usuario actualmente logueado
        currentUser = mAuth.getCurrentUser();
        // Obtener el correo electrónico del usuario
        emailId = currentUser.getEmail();
        // Obtener la referencia al documento del usuario en la colección "users"
        userRef = db.collection("users").document(emailId);
        // Obtener los datos del documento del usuario
        userRef.get().addOnCompleteListener(task -> {
            if (task.isSuccessful()) {
                // Obtener el documento
                document = task.getResult();
                if (document.exists()) {
                    // Obtener los valores de los campos del documento
                    userEmail = document.getString("email");
                    userNombre = document.getString("name");
                    userTelefono = document.getString("phone");
                    // Establecer los valores en los TextView correspondientes
                    if (userEmail != null) {
                        email.setText(userEmail);
                    }
                    if (userNombre != null) {
                        nombre.setText(userNombre);
                    }
                    if (userTelefono != null) {
                        telefono.setText(userTelefono);
                    }
                }
            }
        });
    }
}
```

Editar datos del usuario

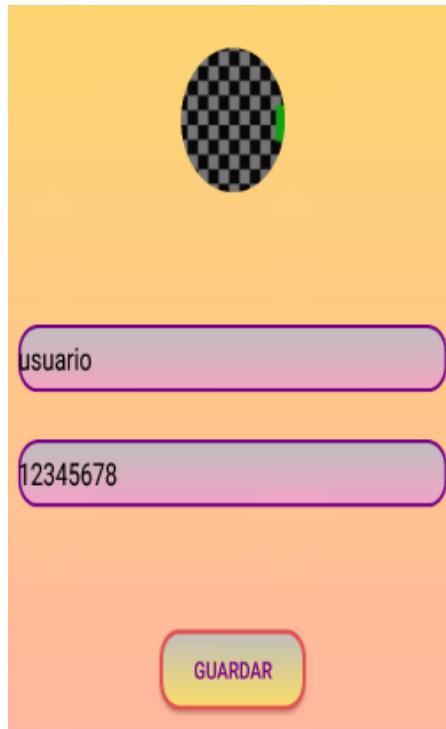
Dentro de esta actividad el usuario podrá modificar el nombre, teléfono y foto.

Se cargan los datos del usuario para facilitar la modificación, si no tiene datos se mostrarán los campos en blanco.

Editar sin datos



Editar con datos



Al seleccionar la imagen dentro de Editar, se mostrará un pop-up para elegir cámara o galería.



El funcionamiento principal de la actividad home y editar es hacer una consulta en firebase para traer los datos del usuario filtrando por el correo.

Además, en Editar se utiliza el método update, donde se manda la colección, el documento (con el correo del usuario) y un hashMap con los datos.

```
// El documento del usuario ya existe, actualizamos sus datos
db.collection( collectionPath: "users").document(email).update(map)
    .addOnSuccessListener(unused -> {
        // Mostramos el mensaje de actualizado
        Toast.makeText( context: EditarDatos.this, text: "Datos actualizados", Toast.LENGTH_SHORT).show();
        homeAv = new Intent( packageContext: EditarDatos.this, HomeActivity.class);
        startActivity(homeAv);
        // Crear el Intent con los datos a enviar
        finish();
    })
    .addOnFailureListener(e -> {
        // Mostramos el mensaje de error
        Toast.makeText( context: EditarDatos.this, text: "Error al actualizar los datos", Toast.LENGTH_SHORT).show();
    });
}
```

El mapa se llenará de los nuevos datos.

```
// Creamos el mapa con los datos a actualizar
Map<String, Object> map = new HashMap<>();
map.put("name", etNombreUsuario.getText().toString());
map.put("phone", etTelefonoUsuario.getText().toString());
// Verificamos si el documento del usuario ya existe en la base de datos
db.collection( collectionPath: "users").document(email).get()
    .addOnCompleteListener(task -> {
        if (task.isSuccessful()) {
            document = task.getResult();
            if (document.exists()) {
                actualizarImagenBd(reference, Uri.parse(uriCapturada));
                // Verificar si no se seleccionó una nueva imagen y conservar el valor existente
                if (uriCapturada.isEmpty()) {
                    map.put("imagen", document.getString( field: "imagen"));
                } else {
                    map.put("imagen", uriCapturada);
                }
            }
        }
    });
}
```

En caso de que el usuario no desee realizar modificaciones, se procede a sobrescribir los datos antiguos (ya cargados) con los nuevos.

Home con datos



Gestión de datos

Dentro del menú, si el usuario está logueado, se encuentra la opción Editar.

Editar

Al entrar se encuentran los botones PERSONAS, RESTAURANTES y PLATOS.

Cada opción permite ver, editar o borrar los datos relacionados al usuario.



Personas

Dentro de esta actividad se hace una consulta dentro de la base de datos, en la colección personas, filtrando por el usuario (whereEqualTo), con esos datos se llena un RecyclerView personalizado donde cada fila es la imagen y el nombre de la persona.

Se muestra una lista de todas las personas que alguna vez el usuario ha añadido.

```
// Obtén la colección "personas" en Firestore
personasRef = db.collection( collectionPath: "personas");
// Realiza la consulta para obtener todas las personas del usuario actual
consulta = personasRef.whereEqualTo( field: "userId", email);
consulta.get().addOnCompleteListener(task -> {
    if (task.isSuccessful()) {
```



El color amarillo representa esta colección, las filas tienen el estilo amarillo grisáceo. Las personas sin imagen, por defecto, son asignados con el logo amarillo. Para el mejor funcionamiento y para evitar la redundancia, al seleccionar una persona se envían a través del intent, un objeto persona y la lista de todas las personas.

```
@Override
public void onItemClickCrudPersona(int position) {
    intentDetalle = new Intent( packageContext: this, DetalleEditarPersonaBd.class);
    intentDetalle.putExtra( name: "comensalDetalle", comensalesBd.get(position));
    intentDetalle.putExtra( name: "comensalesTotales", comensalesBd);
    rLauncherPersonas.launch(intentDetalle);
}
```

Finalmente, se abrirá otra actividad con el detalle de la persona seleccionada y sus datos se podrán modificar desde la nueva actividad.



Dentro de esta actividad se recoge el intent con el objeto persona y la lista de personas para utilizarlos más adelante.

```
comensalBd = (Persona) intent.getSerializableExtra( name: "comensalDetalle");
comensalesTotales = (ArrayList<Persona>) intent.getSerializableExtra( name: "comensalesTotales");
```

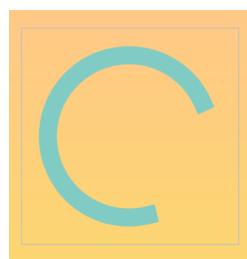
En esta actividad no es necesario hacer una consulta para llenar datos debido a que la actividad *EditarPersona* envía el objeto persona con todos los datos del usuario. Para llenar los datos, se crea un método *mostrarDatos* el cual asigna a los campos nombre, imagen y descripción los datos del objeto recibido.

```
private void mostrarDatos() {  
    etDetalleNombreEditarPersona.setText(comensalBd.getNombre());  
    etDetalleDescripcionEditarPersona.setText(comensalBd.getDescripcion());  
    imagen = comensalBd.getUrlImage();
```

Para la imagen se ha generado un método adicional que muestra una barra de progreso en el caso de que la imagen tarde en cargarse.

```
Glide.with(activity.DetallesEditarPersonaBd.this) .RequestManager  
.load(imagen) .RequestBuilder<Drawable>  
    .listener(new RequestListener<Drawable>() {  
        @Javier Calderón  
        @Override  
        public boolean onLoadFailed(@Nullable GlideException e, Object model, Target<Drawable> target, boolean isFirstResource) {  
            progressBar.setVisibility(View.GONE);  
            return false;  
        }  
        @Javier Calderón  
        @Override  
        public boolean onResourceReady(Drawable resource, Object model, Target<Drawable> target, DataSource dataSource, boolean isFirstResource) {  
            progressBar.setVisibility(View.GONE);  
            return false;  
        }  
    })  
    .circleCrop()  
    .into(ivDetalleFotoEditarPersona);
```

Se mostrará una barra de progreso mientras la imagen se carga.



Cuando termine, se ocultará la barra de progreso y se mostrará la imagen.



En esta actividad se comprueba que el nombre nuevo no existe en la base de datos del usuario, para ello se utiliza la lista de comensales recibida anteriormente y se valida con una variable boolean (siempre estará verdadera), se pondrá en falsa si encuentra un nombre igual dentro de la lista, además si el nombre encontrado es el mismo nombre del objeto recibido, lo ignorará.

```
for (int i = 0; i < comensalesTotales.size(); i++) {
    if (nombreNuevo.equalsIgnoreCase(comensalesTotales.get(i).getNombre())) {
        // Ignorar el elemento "nombre" enviado desde otra actividad
        if (!comensalesTotales.get(i).getNombre().equalsIgnoreCase(comensalBd.getNombre())) {
            comprobarNombre = false;
            break;
        }
    }
}
```

También se validará que el nombre nuevo no esté vacío.

```
if (nombreNuevo.equalsIgnoreCase("")){
    // El campo de texto está vacío, mostrar un mensaje de error
    Toast.makeText(context: DetalleEditarPersonaBd.this, text: "El campo nombre no puede estar vacío, no se realizaron los cambios.", Toast.LENGTH_SHORT).show();
    return;
}
```

El campo nombre no puede estar vacío, no se realizaron los cambios.

En el caso de que no se encuentren errores se realizará una consulta a la colección “personas” filtrando por nombre y el correo del usuario.

```
personasRef = db.collection( collectionPath: "personas");
personasRef.whereEqualTo( field: "nombre", comensalBd.getNombre() ) Query
    .whereEqualTo( field: "usuarioId", email )
    .get() Task<QuerySnapshot>
    .addOnSuccessListener(querySnapshot -> {
```

Finalmente, se realiza un update con los nuevos datos.

```
if (comprobarNombre) {
    document.getReference().update( field: "nombre", nombreNuevo, ...moreFieldsAndValues: "descripcion", descripcionNueva)
        .addOnSuccessListener(aVoid -> {
```

Para las imágenes se ha realizado un método adicional debido a que las imágenes se almacenan en storage, firestore almacena la ruta de la imagen del storage dentro de la colección “personas”.

```
private void subirImagenPersona(String personaId, String imagen) {
    if (personaId != null && !personaId.isEmpty()) {
        // Obtiene una referencia al Storage de Firebase
        storageRef = FirebaseStorage.getInstance().getReference();

        // Define una referencia a la imagen en Storage utilizando el ID de la persona
        rutaImagen = "personas/" + personaId + ".jpg";
        imagenRef = storageRef.child(rutaImagen);

        // Sube la imagen a Storage
        imagenUri = Uri.parse(imagen);
        uploadTask = imagenRef.putFile(imagenUri);

        uploadTask.addOnSuccessListener(taskSnapshot -> {
            // La imagen se subió exitosamente
            // Obtiene la URL de descarga de la imagen
            imagenRef.getDownloadUrl().addOnSuccessListener(uri -> {
                // La URL de descarga de la imagen está disponible
                imageUrl = uri.toString();
                // Actualiza el campo "imagen" en Firestore con la URL de descarga de la imagen
                db.collection( collectionPath: "personas").document(personaId) DocumentReference
                    .update( field: "imagen", imageUrl) Task<Void>
```

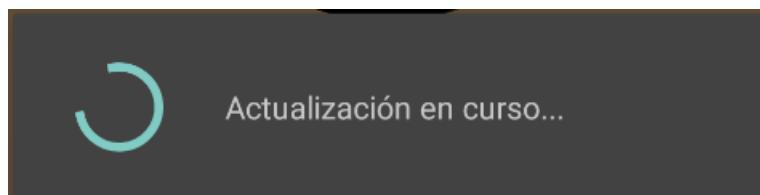
En este método se hace referencia dentro de storage a la carpeta “personas” y después se realiza un update en firestore, donde se envía por parámetro la nueva imagen.

Al seleccionar “*EDITAR*” se realizan todas las comprobaciones anteriores y se hace una llamada al método de editar donde se realiza dentro un update con los nuevos datos.



Por último, se añade un cuadro de texto de espera antes de finalizar la actividad. Si no se retarda este proceso, debido a la velocidad a la que se produce todo el cambio, se actualiza dentro de la base de datos la nueva persona, pero no se verán aplicados los nuevos cambios en la lista.

Para evitar ese problema se ha añadido el cuadro de texto y dos segundos de espera.



```
//Botón que vuelve a comensales y además devuelve el comensal modificado
btnEditarDetalleEditarPersona.setOnClickListener(view -> {
    editarComensal();
    //espero dos segundos para que se realicen bien los cambios
    progressDialog = ProgressDialog.show( context: this, title: "", message: "Actualización en curso...", indeterminate: true);
    handler = new Handler();
    handler.postDelayed(() -> {
        // Quitar el ProgressDialog después de 500 milisegundos adicionales
        if (progressDialog != null && progressDialog.isShowing()) {
            progressDialog.dismiss();
        }
        finish();
    }, delayMillis: 2000);
});
```

Al terminar, se finaliza la actividad actual, volviendo atrás con la nueva lista.

Dentro de la actividad *Detalle* también se encuentra el botón de borrar.



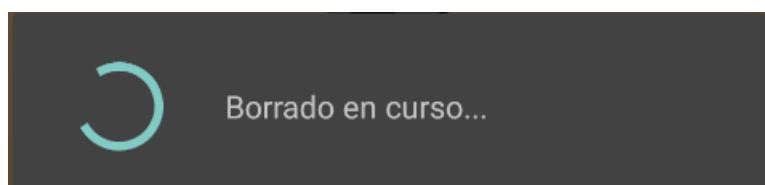
Al seleccionar este botón, se realiza una consulta en firestore dentro de la colección personas filtrando por nombre y correo. Si encuentra algún campo que coincida con el nombre y correo proporcionados en el detalle dentro de la base de datos se procede al borrado.

```
personasRef.whereEqualTo("nombre", comensalBd.getNombre())
    .whereEqualTo("userId", email)
    .get()
    .addOnSuccessListener(querySnapshot -> {
        if (!querySnapshot.isEmpty()) {
            document = querySnapshot.getDocuments().get(0);
            documentgetReference().delete()
                .addOnSuccessListener(aVoid -> {
                    // La eliminación se realizó exitosamente
                })
                .addOnFailureListener(e -> {
                    // Ocurrió un error al eliminar el comensal
                    Toast.makeText(context, "Error al eliminar el comensal", Toast.LENGTH_SHORT).show();
                });
        }
    });
}
```

Se produce el mismo proceso que en editar, el programa realiza los cambios, pero aparece un cuadro de texto con 2 segundos de espera para así poderse actualizar también la lista.

```
//Botón de borrar, llama a un método que a además de borrar
btnBorrarDetalleEditarPersona.setOnClickListener(view -> {
    borrarComensal();
    progressDialog = ProgressDialog.show(this, "", "Borrado en curso...", indeterminate: true);
    handler = new Handler();
    handler.postDelayed(() -> {
        // Quitar el ProgressDialog después de 500 milisegundos adicionales
        if (progressDialog != null && progressDialog.isShowing()) {
            progressDialog.dismiss();
        }
        finish();
    }, delayMillis: 2000);
});
```

Al terminar, se finaliza la actividad actual, volviendo atrás con la nueva lista.



Restaurantes

Dentro de esta actividad se hace una consulta dentro de la base de datos, en la colección restaurantes, filtrando por el usuario (whereEqualTo), con esos datos se llena un RecyclerView personalizado donde cada fila es el nombre del restaurante.

Se muestra una lista de todos los restaurantes que alguna vez el usuario ha guardado.

```
if (currentUser != null) { // Verifica si hay un usuario actualmente logueado
    restaurantesBd = new ArrayList<>(); // Crea una nueva lista para almacenar los restaurantes
    email = currentUser.getEmail(); // Obtiene el correo electrónico del usuario actual

    // Configura la referencia a la colección "restaurantes" en la base de datos
    restaurantesRef = db.collection(collectionPath: "restaurantes");

    // Realiza una consulta a la colección "restaurantes" donde el campo "userId" sea igual al correo electrónico del usuario actual
    consulta = restaurantesRef.whereEqualTo(field: "userId", email);

    // Ejecuta la consulta y agrega un listener para recibir el resultado
    consulta.get().addOnCompleteListener(task -> {
```



El color azul representa esta colección y cada fila tiene el color azul grisáceo.

Para el mejor funcionamiento y para evitar la redundancia, al seleccionar un restaurante se envían a través del intent, un objeto restaurante y la lista de todos los restaurantes.

```
• Relyak
@Override
public void onItemClick(int position) {
    intent = new Intent( packageContext: this, DetalleEditarRestaurantesBd.class);
    intent.putExtra( name: "restauranteDetalle", restaurantesBd.get(position));
    intent.putExtra( name: "restaurantesTotales", restaurantesBd);
    rLauncherRestaurantes.launch(intent);
}
```

Finalmente, se abrirá otra actividad con el detalle del restaurante seleccionado y sus datos se podrán modificar desde la nueva actividad.



Con el intent recogido se carga el nombre del restaurante.

En esta clase se comprueba que el nuevo nombre no esté vacío ni tampoco repetido que el nombre ya exista en la lista.

```

for (int i = 0; i < restaurantesTotales.size(); i++) {
    if (nombreNuevo.equalsIgnoreCase(restaurantesTotales.get(i).getNombreRestaurante())) {
        // Ignorar el elemento "nombre" enviado desde otra actividad
        if (!restaurantesTotales.get(i).getNombreRestaurante().equalsIgnoreCase(restaurantesBd.getNombreRestaurante())) {
            comprobarNombre = false;
            break;
        }
    }
}

if (nombreNuevo.isEmpty()) {
    // El campo de texto está vacío, mostrar un mensaje de error
    Toast.makeText(contexto, DetalleEditarRestaurantesBd.this, text: "El campo nombre de restaurante no puede estar vacío, no se realizaron los cambios.", Toast.LENGTH_SHORT).show();
    return;
}

```

Si se cumplen las comprobaciones, se procederá a actualizar los restaurantes utilizando el método update.

Esta clase tiene un funcionamiento distinto debido a que los platos tienen un restaurante de referencia, firestore no es relacional y no actualizará los campos restaurantes dentro de los platos

Para solventar ese problema al actualizar el restaurante se hace una consulta dentro de la colección platos filtrando por nombre antiguo y por email del usuario, a posteriori se actualizan los campos con el nuevo nombre.

```

private void actualizarReferencias(String nombreAntiguo, String nombreNuevo) {
    // Obtener la referencia a la colección "Platos"
    platosRef = db.collection(collectionPath: "platos");
    // Realizar una consulta para obtener los platos con el nombre antiguo
    platosRef.whereEqualTo(field: "restaurante", nombreAntiguo).whereEqualTo(field: "usuario", currentUser.getEmail()).get().addOnSuccessListener(querySnapshot -> {
        if (!querySnapshot.isEmpty()) {
            // Actualizar cada documento que coincida con el nombre antiguo
            for (DocumentSnapshot document : querySnapshot.getDocuments()) {
                document.getReference().update(field: "restaurante", nombreNuevo).addOnSuccessListener(aVoid -> {
                    // La actualización se realizó exitosamente para el documento actual
                }).addOnFailureListener(e -> {
                    // Ocurrió un error al actualizar el documento
                });
            }
        }
        .addOnFailureListener(e -> {
            // Ocurrió un error al realizar la consulta
        });
    });
}

```

Para el borrado funciona exactamente igual, si el restaurante no existe y no se borran esas referencias es imposible acceder a esos platos.

Se ha implementado un método que, al borrar el restaurante, busque los platos con ese nombre y los borre junto con el restaurante.

```
private void borrarReferencias(String nombreRestaurante) {
    // Obtener la referencia a la colección "Platos"
    platosRef = db.collection("platos");
    // Realizar una consulta para obtener los platos con el nombre del restaurante
    platosRef.whereEqualTo("restaurante", nombreRestaurante).whereEqualTo("usuario", currentUser.getEmail()).get().addOnSuccessListener(querySnapshot -> {
        if (!querySnapshot.isEmpty()) {
            // Eliminar cada documento que coincida con el nombre del restaurante
            for (DocumentSnapshot document : querySnapshot.getDocuments()) {
                document.getReference().delete().addOnSuccessListener(aVoid -> {
                    // La eliminación se realizó exitosamente para el documento actual
                }).addOnFailureListener(e -> {
                    // Ocurrió un error al eliminar el documento
                });
            }
        }
    }).addOnFailureListener(e -> {
        // Ocurrió un error al realizar la consulta
    });
}
```

Ambos métodos, al terminar, esperarán 2 segundos.

```
btnEditarDetalleRestaurantes.setOnClickListener(view -> {
    //espero dos segundos para que se realicen bien los cambios
    editarRestaurante();
    progressDialog = ProgressDialog.show(context, title "", message: "Actualización en curso...", indeterminate: true);

btnBorrarDetalleRestaurantes.setOnClickListener(view -> {
    borrarRestaurante();
    progressDialog = ProgressDialog.show(context, title "", message: "Borrado en curso...", indeterminate: true);

handler = new Handler();
handler.postDelayed(() -> {
    // Quitar el ProgressDialog después de 2 segundos adicionales
    if (progressDialog != null && progressDialog.isShowing()) {
        progressDialog.dismiss();
    }
    finish();
}, delayMillis: 2000);
```

Al terminar, se finaliza la actividad actual, volviendo atrás con la nueva lista.

Platos

Dentro de esta actividad se hace una consulta dentro de la base de datos, en la colección platos, filtrando por el usuario (whereEqualTo), con esos datos se llena un RecyclerView personalizado donde cada fila es el nombre del plato y su imagen. Se muestra una lista de todos los platos que alguna vez el usuario ha añadido.



El color morado representa esta colección, las filas tienen el estilo morado grisáceo. Los platos sin imagen, por defecto, son asignados con el logo morado. Para el mejor funcionamiento y para evitar la redundancia, al seleccionar un plato, se envían a través del intent, un objeto plato y la lista de todos los platos.

```
@Override
public void onItemClick(int position) {
    intentDetalle = new Intent( packageContext: this, DetalleEditarPlatosBd.class);
    intentDetalle.putExtra( name: "platoDetalle", platosBd.get(position));
    intentDetalle.putExtra( name: "platosTotales", platosBd);
    rLauncherPlatos.launch(intentDetalle);
}
```

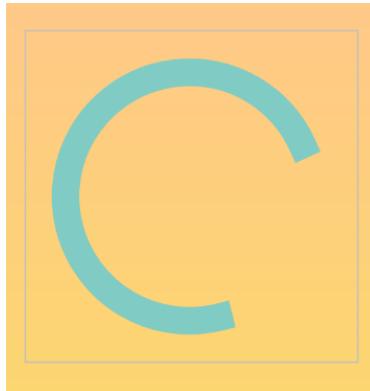
Finalmente, se abrirá otra actividad con el detalle del plato seleccionado y sus datos se podrán modificar desde la nueva actividad.



Con el intent recogido se asignan los datos al plato, el nombre del restaurante no se podrá modificar.

Para la imagen se ha generado un método adicional que muestra una barra de progreso en el caso de que la imagen tarde en cargarse.

Sin cargar



Cargado



Como en las otras actividades, se comprobará que el nombre del nuevo plato no coincida con el usuario, plato y restaurante de la base de datos.

El usuario podrá tener varios platos con el mismo nombre mientras sean de distintos restaurantes.

```
// Comprobación del nombre y el restaurante del plato a editar
for (int i = 0; i < platosTotales.size(); i++) {
    nombreExistente = platosTotales.get(i).getNombre();
    restauranteExistente = platosTotales.get(i).getRestaurante();

    // Ignorar el elemento "nombre" enviado desde otra actividad
    if (!nombreExistente.equalsIgnoreCase(platoBd.getNombre()) ||
        !restauranteExistente.equalsIgnoreCase(platoBd.getRestaurante())) {
        // Verificar si el nombre y el restaurante coinciden con otro plato
        if (nombreNuevo.equalsIgnoreCase(nombreExistente) &
            restauranteExistente.equalsIgnoreCase(platoBd.getRestaurante())) {
            comprobarNombre = false;
            break;
        }
    }
}
```

Se comprobará que el nombre no esté vacío y se hará una consulta a la base de datos filtrando por usuario, restaurante y nombre.

```

platosRef = db.collection("collectionName: platos");
platosRef.whereEqualTo("nombre", platoBd.getNombre()) Query
    .whereEqualTo("restaurante", platoBd.getRestaurante())
    .whereEqualTo("usuario", email)
    .get() Task<QuerySnapshot>
    .addOnSuccessListener(querySnapshot -> {
        if (!querySnapshot.isEmpty()) {
            document = querySnapshot.getDocuments().get(0);
            if (comprobarNombre) {
                document.getReference().update("nombre", nombreNuevo, ...moreFieldsAndValues: "descripcion", descripcionNueva, "precio", precioNuevo)
            }
        }
    });
}

```

A continuación, se realizará el update en caso de seleccionar la opción editar.

Para la imagen se abrirá la cámara o galería, y llamará al método para insertar y actualizar la imagen dentro de storage.

```

// Obtén una referencia al Storage de Firebase
storageRef = FirebaseStorage.getInstance().getReference();

// Define una referencia a la imagen en Storage utilizando el ID del plato
rutaImagen = "platos/" + platoId + ".jpg";
imagenRef = storageRef.child(rutaImagen);

// Sube la imagen a Storage
imagenUri = Uri.parse(imagen);
uploadTask = imagenRef.putFile(imagenUri);

```

Si se cumplen todas las validaciones se procederá a actualizar el plato con los nuevos datos. Se esperará dos segundos para terminar la actividad y evitar errores.

El método de borrado funciona igual que en las clases anteriores, filtra por nombre, usuario y restaurante y borra el plato seleccionado.

```

// Eliminar el plato de la base de datos
platosRef = db.collection("collectionPath: "platos");

platosRef.whereEqualTo("nombre", platoBd.getNombre()) Query
    .whereEqualTo("restaurante", platoBd.getRestaurante())
    .whereEqualTo("usuario", email)
    .get() Task<QuerySnapshot>
    .addOnSuccessListener(querySnapshot -> {
        if (!querySnapshot.isEmpty()) {
            document = querySnapshot.getDocuments().get(0);
            document.getReference().delete()
        }
    });
}

```

Al igual que actualizar, se esperará dos segundos antes de terminar la actividad.

Participantes



A partir de esta actividad podemos gestionar los comensales, podemos **añadir** comensales, **editarlos**, etc.

Visualmente podemos comprobar que se trata de una lista que muestra **2 objetos** por cada fila, aportando un diseño más dinámico y flexible a la hora de mostrar grandes cantidades de comensales.

Es realmente importante identificar que el primer elemento de la lista se trata de un **propio** objeto 'Comensal' pero con aspecto visual y **funcionamiento distinto**.

(LOGUEADO)

En el caso de estar logueado esta lista de presenta todas las personas que anteriormente añadió el usuario, mejorando considerablemente la velocidad para completar una cuenta y haciendo mucho más sencillo.

Añadir persona



Objeto de la lista que simula ser un botón, con el método OnClick podemos gestionar su funcionamiento, en este caso la posición 0 siempre será el botón Añadir Persona y dará inicio a la actividad para añadir participante.

Editar persona



Objeto de la lista Comensales, a medida que añades más personas va aumentando la lista.

Al pulsar un objeto de la lista (persona) puedes acceder a la actividad para editar persona, que te da la opción de modificar y/o añadir más platos, dando al usuario la opción de gestionar los elementos de manera fácil y sencilla.

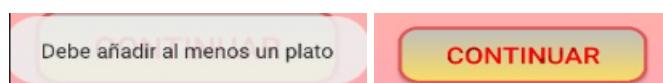
(LOGUEADO)

Al añadir un participante, si el usuario está logueado, automáticamente se añadirá una persona a la colección de la base de datos, al abrir otra cuenta, se presentará en una lista horizontal cada persona que añadida, agilizando el proceso considerablemente en futuras cuentas.



Continuar

Botón que comprueba si se ha añadido al menos un comensal y un plato, en el caso de que negativo, no funcionará y mostrará un aviso.



Si se ha añadido un comensal y al menos un plato, dará acceso a la actividad DesgloseActivity que muestra el resultado final de reparto.

Esta clase cuenta con los métodos generales para asignar IDs a las variables, efectos a los elementos y mostrar la lista de comensales

```
//Método que asigna IDs a los elementos
asignarId();

//Método que asigna los efectos a los elementos
asignarEfectos();

//Método que muestra el contenido del adaptador
mostrarAdapter();
```

OnItemClick

Método que gestiona la lista de participantes, dependiendo que posición se pulsa hará una acción u otra, es importante aclarar que, se utilizan variables ActivityResultLauncher para iniciar actividades hijas, las posibles acciones son las siguientes:

```
public void onItemClick(int position) {
    comensalPosicion = position;
    //Si pulsas "Añadir Persona" ( 0 ), accederás a la actividad añadir persona
    if (position > 0) {
        //Accede a la actividad detalle de una persona
        intent = new Intent( packageContext: this, DetallePersonaActivity.class);
        intent.putExtra( name: "comensalDetalle", comensales.get(position));
        intent.putExtra( name: "arrayListComenComp", comensales);
        intent.putExtra( name: "nombreRestaurante", nombreRestaurante);
        \LauncherDetalleComensal.launch(intent);
    } else {
        //Accede a la actividad para añadir nuevos comensales
        Intent intent = new Intent( packageContext: this, AnadirParticipanteActivity.class);
        intent.putExtra( name: "arrayListComensales", comensales);
        intent.putExtra( name: "nombreRestaurante", nombreRestaurante);
        \LauncherAnadirComensal.launch(intent);
    }
}
```

Al pulsar la posición 0 accederás a la actividad AñadirPersonaActivity, esta actividad espera un ArrayList de comensales para poder añadir el nuevo comensal y el nombreRestaurante para poder hacer peticiones a la base de datos y (**LOGUEADO**) obtener los platos añadidos anteriormente de este restaurante en concreto.

Al pulsar otros elementos de la lista, se accede a DetallePersonaActivity, que muestra el detalle de un comensal en concreto, recibe el comensal seleccionado a través de un .get(position), de esta manera se edita el comensal en concreto, además, espera recibir una lista completa de comensales, utilizada para compartir un plato en concreto y finalmente el nombre de restaurante.

(LOGUEADO)

Al cerrar las actividades hijas abiertas utilizando ActivityResultLauncher se da paso a las siguientes acciones.

```
//Launcher Result recibe el ArrayList con los nuevos comensales y los inserta en el adapter
rLauncherAnadirComensal = registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(), result -> {
        if (result.getResultCode() == Activity.RESULT_OK) {
            Intent data = result.getData();
            comensales = (ArrayList<Persona>) data.getSerializableExtra( name: "arrayListComensales");
            personaAdapter.setResultsPersona(comensales);
        }
    });
};
```

Al cerrar la actividad hija AnadirParticipanteActivity devuelve a la actividad padre (ParticipantesActivity) la lista de comensales con el nuevo comensal añadido, tras recibirla, se inserta el nuevo ArrayList comensales en el adapter para **actualizar la lista**.

```
rLauncherDetalleComensal = registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(), result -> {
        if (result.getResultCode() == Activity.RESULT_OK) {
            comprobarLauncher();
            intent = result.getData();
            borrarComensal = intent.getBooleanExtra( name: "borrarComensal", defaultValue: false);
            if (!borrarComensal) {
                comensales.set(comensalPosicion, (Persona) intent.getSerializableExtra( name: "detalleComensal"));
                personaAdapter.setResultsPersona(comensales);
            } else {
                //Método que comprueba con quien has compartido y lo borra de ser necesario
                borrarCompartidos();
                comensales.remove(comensalPosicion);
                personaAdapter.setResultsPersona(comensales);
                //Importantísimo, reorganiza los ComensalCodes al borrar un comensal
                for (int i = 1; i < comensales.size(); i++) {
                    comensales.get(i).setComensalCode(i);
                }
            }
        } else {
            comprobarLauncher();
        }
    });
};
```

En el caso de DetallePersonaActivity el usuario tiene la opción de editar o borrar un comensal, a través de la variable “borrarComensal” gestionamos su función.

Si la función es editar, comensales remplaza la posición del comensal seleccionado por el nuevo comensal editado, utilizamos una variable comensalPosition para almacenar la posición del comensal elegido anteriormente, acto seguido se actualizará la lista.

En el caso de haber elegido borrar, se borra el comensal elegido anteriormente, utilizando comensalPosition.

Es importante resaltar que, se debe realizar un reajuste de cada persona, ya que cada comensal tiene un atributo int comensalCode, que funciona en base al tamaño del ArrayList y diferencia cada uno de los comensales, aunque estos se llamen igual.

Finalmente, se accede al método borrarCompartidos(), método extremadamente complejo que comprueba si algún comensal ha compartido un plato **con el usuario borrado**, en el caso de que se haya borrado una persona con un plato compartido, **dicho plato será borrado**.

```
public void borrarCompartidos() {  
    for (int i = 1; i < comensales.size(); i++) {  
        for (int j = 1; j < comensales.get(i).getPlatos().size(); j++) {  
            if (comensales.get(i).getPlatos().get(j).isCompartido()) {  
                for (int h = 0; h < comensales.get(i).getPlatos().get(j).getPersonasCompartir().size(); h++) {  
                    if (comensales.get(i).getPlatos().get(j).getPersonasCompartir().get(h).getComensalCode() == comensales.get(comensalPosicion).getComensalCode()) {  
                        comensales.get(i).getPlatos().remove(j);  
                        break;  
                    }  
                }  
            }  
        }  
    }  
}
```

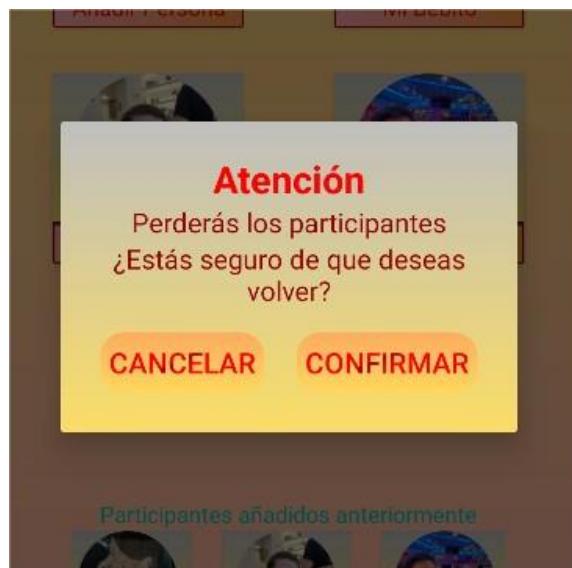
Se recorren todos los comensales, sus platos y en cada plato las personas compartidas.

Popup volver ATENCIÓN

Se ha creado un pop-up preventivo con el fin de evitar que el usuario abandone la lista de comensales y pierda la información de forma accidental

Este esperará una confirmación por parte del usuario, evitando accidentes incómodos o pérdida de tiempo, mejorando la usabilidad y la experiencia para el usuario

Dicho pop-up iniciará “volver” utilizando el botón nativo Android.



Al confirmar que se desea abandonar la actividad **Participantes**, es importante destacar que es recomendable cerrar el popup, utilizando `.dismiss()` antes de cerrar la actividad, al no hacerlo produce fugas de memoria y algunos errores visuales molestos.

Se borrarán todos los comensales y se volverá a **IndexActivity** para empezar de nuevo o realizar otras gestiones.

```
//Botones para el popup de confirmación
//Confirmar, retrocede, cierra la actividad y pierde los datos introducidos - se debe cerrar con dismiss() para evitar fugas de memoria
btnConfirmarParticipantes.setOnClickListener(view -> {
    startActivity(volverIndex);
    puVolverParticipantes.dismiss();
    finish();
});
```

Al elegir Cancelar o pulsar fuera del pop-up, este desaparecerá sin realizar ningún cambio

```
//Cancela, desaparece el popup y continúa en la actividad
btnCancelarParticipantes.setOnClickListener(view -> puVolverParticipantes.dismiss());
```

Al pulsar el botón **Continuar** se realizarán varias comprobaciones antes de pasar al **DesgloseActivity**, se debe revisar que haya al menos un comensal y un plato, de cumplir todas las condiciones, avanzará a **DesgloseActivity** enviando la lista de comensales completa con todos los comensales y sus platos, en **Desglose** se realizarán las operaciones para calcular el resultado.

```
//Botón de continuar, avanza a la siguiente actividad
// 1 - Debe haber mínimo un comensal
// 2 - Debe haber mínimo un plato ( la primera vez )
btnContinuarParticipantes.setOnClickListener(view -> {
    //Obtenemos todos los platos
    int numPlatos = obtenerPlatos();
    if (comensales.size() > 1 && numPlatos >= MINIMO_PLATOS) {
        //Accede a la actividad detalle de una persona
        intent = new Intent(packageContext: this, DesgloseActivity.class);
        intent.putExtra(name: "envioDesglose", comensales);
        intent.putExtra(name: "nombreRestaurante", nombreRestaurante);
        rLauncherDesglose.launch(intent);
    } else {
        //Comprobamos qué elemento nos falta para avanzar al desglose
        if (comensales.size() < 2) {
            Toast.makeText(context: this, text: "Debe añadir al menos un comensal", Toast.LENGTH_SHORT).show();
        }
        if (numPlatos < MINIMO_PLATOS) {
            Toast.makeText(context: this, text: "Debe añadir al menos un plato", Toast.LENGTH_SHORT).show();
        }
    }
});
```

cargarDatosBD (LOGUEADO)

En el caso de estar logueado se accederá al método que obtiene el email del usuario y la colección personas para poder realizar las nuevas peticiones y recibir las personas almacenadas en la base de datos únicamente del usuario logueado, los usuarios descargados de la base de datos se almacenan en el ArrayList comensalesBD y se muestran en la lista horizontal.

```
private void cargarDatosBd() {
    if (currentUser != null) {
        comensalesBd = new ArrayList<>();
        email = currentUser.getEmail(); // Utiliza el email como ID Único del usuario
        // Obtén la colección "personas" en Firestore
        personasRef = db.collection( collectionPath: "personas");
        // Realiza la consulta para obtener todas las personas del usuario actual
        consulta = personasRef.whereEqualTo( field: "userId", email);
        consulta.get().addOnCompleteListener(task -> {
            if (task.isSuccessful()) {
                // Recorrer los documentos obtenidos y agregar los datos al ArrayList
                for (DocumentSnapshot document : task.getResult()) {
                    nombre = document.getString( field: "nombre");
                    descripcion = document.getString( field: "descripcion");
                    imagen = document.getString( field: "imagen");
                    persona = new Persona(nombre, descripcion, imagen);
                    comensalesBd.add(persona);
                }
                // Notificar al adapter que los datos han cambiado
                personaAdapterBd.setResultsPersonaBd(comensalesBd);
            }
        });
    }
}
```

onItemClickBD (LOGUEADO)

Al pulsar un objeto Persona de la lista horizontal, esta enviará a la actividad AnadirParticipanteActivity el comensal descargado y lo insertará agilizando el proceso de añadir comensales a la cuenta.

```
@Override
public void onItemClickBd(int position) {
    Intent i = new Intent( packageContext: this, AnadirParticipanteActivity.class);
    i.putExtra( name: "importado", value: true);
    i.putExtra( name: "arrayListComensales", comensales);
    i.putExtra( name: "comensalesBd", comensalesBd.get(position));
    i.putExtra( name: "nombreRestaurante", nombreRestaurante);
    rLauncherAnadirComensal.launch(i);
}
```

Añadir participante

Desde esta actividad podemos generar un participante para la cuenta, es obligatorio añadir un nombre.

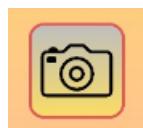


La actividad ofrece ya la opción de añadir platos, se pueden comprobar por su correspondiente color

Siguiendo la temática anterior, el ArrayList de platos cuenta con un objeto posición 0 utilizado para añadir plato, pulsando los elementos plato accederás a DetallePlatoActivity.

El usuario tiene la opción de añadir una imagen a la persona que piensa añadir, si bien no es obligatorio, realmente ayuda bastante para identificar de la lista de personas.

Se utiliza un ImageView con el método OnClickListener permitiendo que el ImageView se comporte como un botón.



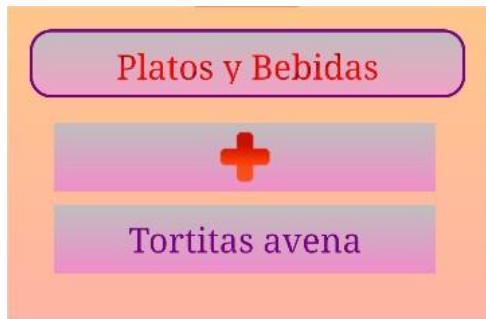
Al pulsar el ícono de la cámara se mostrará un pop-up ofreciendo al usuario la opción de abrir la cámara para una imagen nueva o la galería para elegir una imagen ya almacenada.



Para ambas opciones se pedirán permisos para poder acceder a los recursos, en caso de negarte, mostrará un aviso de que se requiere activarlos.

Añadir platos

Tras el TextView que indica que posteriormente va la sección de platos, se presenta una lista de comidas única para la persona, cada persona tiene su ArrayList de platos similar a la lista de personas, posición 0 (+) accede a AnadirPlatoActivity, el resto de opciones a DetallePlatoActivity.



El botón continuar tras validar todo añade el comensal a la lista de comensales, tenga platos añadidos o no



No existe una limitación a la hora de añadir platos pues es una lista con la opción de realizar scroll hacia arriba y abajo.

Los platos que tengan un borde celeste son aquellos platos que han sido compartidos con otros participantes en la cuenta, dicho plato compartido solo aparecerá en el comensal que lo ha añadido, pero a la hora de repartir el precio, se repartirá entre los comensales compartidos.



La clase AnadirParticipanteActivity cuenta con los métodos generales, además, recibe el ArrayList de comensales para poder añadir el nuevo comensal.

Se debe destacar la variable boolean importado, esta recibe del Intent un boolean que indica si se importa un elemento de la base de datos o no.

```
//Recibe la lista de comensales para empezar a añadir
intent = getIntent();
comensales = (ArrayList<Persona>) intent.getSerializableExtra( name: "arrayListComensales");
importado = intent.getBooleanExtra( name: "importado", defaultValue: false);
nombreRestaurante = intent.getStringExtra( name: "nombreRestaurante");

//Método que asigna IDs a los elementos
asignarId();

//Método que asigna los efectos a los elementos
asignarEfectos();

//Comprobar usuario si está logueado o no
comprobarLauncher();

//Método que muestra el contenido del adaptador
mostrarAdapter();

//Trae desde el clic de personas ya añadidas recientemente los participantes para añadirlos
insertarDesdeBd();
```

insertarDesdeBd

Utilizamos la variable para comprobar si es un participante nuevo o es un participante importado de la base de datos, ya que la actividad AnadirParticipanteActivity se utiliza para añadir nuevos comensales o descargarlos de la Base de datos.

A través de la variable importado, cambiamos su funcionamiento, en el caso de que sea importado, se añadirá a los EditText el nombre, apellido.

```
if (importado) {
    progressBar.setVisibility(View.VISIBLE);
    personaBd = (Persona) intent.getSerializableExtra( name: "comensalesBd");
    etNombreAnadirP.setText(personaBd.getNombre());
    etDescAnadirP.setText(personaBd.getDescripcion());
```

En el caso de no importar nada desde la base de datos, **NO SE ACCEDERÁ** al método insertarDesdeBd.

A parte de añadirse el nombre y la descripción, si el usuario tenía una imagen almacenada, se cargará utilizando la librería Glide dicha librería aporta distintos métodos extremadamente útiles que mejoran la experiencia del usuario.

```
Glide.with( activity: this ) RequestManager
    .load(personaBd.getUrlImage()) RequestBuilder<Drawable>
        .circleCrop()
        .error(uriCapturada) //Imagen que se añade si hay un error al cargar la imagen
        .listener(new RequestListener<Drawable>() {
            @Javier Calderón
            @Override
            public boolean onLoadFailed(@Nullable GlideException e, Object model, Target<D
                progressBar.setVisibility(View.GONE);
                return false;
            }
            @Javier Calderón
            @Override
            public boolean onResourceReady(Drawable resource, Object model, Target<Drawabl
                progressBar.setVisibility(View.GONE);
                return false;
            }
        })
        .into(ivPlatoAnadirP);
uriCapturada = personaBd.getUrlImage();
```

Glide permite insertar imágenes utilizando tanto URIS como rutas HTTPS permitiendo una flexibilidad inmejorable, además, ofrece los siguientes métodos:

- load(RUTA IMAGEN) Inserta imágenes utilizando distintos tipos de ruta
- circleCrop() Inserta la imagen con un redondeado mejorando su apariencia
- error() En el caso de fallar a la hora de insertar una imagen, inserta una por defecto
- listener() Ofrece opciones en el caso de fallar una descarga o ejecutar elementos hasta que haya cargado la imagen

En este caso el Listener se utiliza junto a un ProgressBar que se mostrará tanto al fallar como hasta tener preparado el recurso, por ejemplo, al descargar una imagen (onResourceReady) mientras se descarga mostrará un progressbar



Finalmente, tras cargar el comensal de la base de datos, se guardará la ruta de la imagen (HTTPS) en la variable uriCapturada

ActivityResultLauncher utilizados en AnadirParticipante

AnadirParticipante al pulsar + de la lista de platos accede a RecordarPlatoActivity, una actividad que muestra platos que hayas añadido anteriormente, desde la base de datos y de la propia cuenta creada.

En el caso de no recordar nada, se accede a la actividad AnadirPlatoActivity en la cual puedes añadir platos nuevos, por el lado contrario, si seleccionas un plato creado anteriormente, se añadirá directamente, agilizando bastante el proceso

```
//Launcher que da la opción a recordar platos o añadir nuevos
rlauncherRecordarPlato = registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(), result -> {
        if (result.getResultCode() == Activity.RESULT_OK) {
            Intent data = result.getData();
            nuevoPlato = data.getBooleanExtra("name: "nuevoPlato", defaultValue: false);
            if (nuevoPlato) {
                //Abre la actividad para añadir nuevo plato
                intent = new Intent(packageContext: this, AnadirPlatoActivity.class);
                intent.putExtra("name: arrayListPlatos", platos);
                intent.putExtra("name: arrayListComenComp", comensales);
                rlauncherPlatos.launch(intent);
            } else {
                //recibe el plato que anteriormente se había utilizado
                platos.add(Plato) data.getSerializableExtra("name: "recordarNuevo"));
                anadirPAdapter.setResultsPlato(platos);
            }
        }
    });
});
```

Similar a DetallePersonaActivity tras haber editado un plato de la lista desde DetallePlatoActivity esta devuelve un boolean que indica si se debe borrar o editar.

En el caso de editar sustituye el plato utilizando la variable platoPosicion para sustituir el correcto y por el lado de borrar, elimina el plato utilizando la variable platoPosicion

```
//Launcher detallePlato se actualiza al recibir la modificación
rLauncherDetallePlato = registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(), result -> {
        if (result.getResultCode() == Activity.RESULT_OK) {
            Intent data = result.getData();
            borrarPlato = data.getBooleanExtra( name: "borrarPlato", defaultValue: false);
            if (!borrarPlato) {
                //Sustituye el plato de la posición elegida anteriormente ( EDITAR PLATO )
                platos.set(platoPosicion, (Plato) data.getSerializableExtra( name: "detallePlato"));
                anadirPAdapter.setResultsPlato(platos);
            } else {
                //Borra el plato de la posición elegida anteriormente ( BORRAR PLATO )
                platos.remove(platoPosicion);
                anadirPAdapter.setResultsPlato(platos);
            }
        }
    });
});
```

Recibe la lista de platos con el nuevo plato ya añadido y actualiza la lista de platos para que aparezca el nuevo plato.

```
//Launcher Result - recibe los platos del usuario creado
rLauncherPlatos = registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(), result -> {
        if (result.getResultCode() == Activity.RESULT_OK) {
            intent = result.getData();
            platos = (ArrayList<Plato>) intent.getSerializableExtra( name: "arrayListPlatos");
            anadirPAdapter.setResultsPlato(platos);
            comprobarLauncher();
        }
    });
});
```

Cámara y Galería

La cámara y galería se muestran a través de un pop-up, dicho pop-up habilita ambos botones para elegir cual método prefieres para asignar una imagen.

Hacer pulsar el botón de la cámara, lo primero que hace es revisar si tiene los permisos para acceder a la cámara, de tener los permisos accede al método abrirCamara.

```
//Añadimos onClick en el ImageView para activar la imagen
btnUpCamara.setOnClickListener(view -> {
    // Solicitar permiso para acceder a la cámara
    if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.CAMERA) == PackageManager.PERMISSION_DENIED) {
        //Si no tenemos los permisos los obtenemos
        ActivityCompat.requestPermissions( activity: this, new String[]{Manifest.permission.CAMERA}, CAMERA_PERMISSION_CODE);
    } else {
        // Si ya se tienen los permisos, abrir la cámara
        abrirCamara();
    }
});
```

En el caso de no tenerlos, carga un popup es nativo de Android para preguntarte si tienes los privilegios, de no aceptarlo aparecerá un mensaje pidiendo activarlos, en el caso de activarlos accedes a abrirCamara.

```
// Método para manejar la respuesta de la solicitud de permisos
▲ Zharell +1
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == CAMERA_PERMISSION_CODE) {
        if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            // Si se conceden los permisos, abrir la cámara
            abrirCamara();
        } else {
            // Si se deniegan los permisos, mostrar un mensaje al usuario
            Toast.makeText(context, "Para almacenar la imagen debe otorgar los permisos", Toast.LENGTH_SHORT).show();
        }
    }
}
```

Método muy sencillo que utiliza una variable ActivityResultLauncher para poder obtener tras realizar la foto, una imagen.

```
private void abrirCamara() {
    intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    cameraLauncher.launch(intent);
}
```

ActivityResultLauncher totalmente importante, este crea un objeto Bitmap para añadir la imagen tomada a la **galería** obteniendo una URI con la cual podemos utilizar para insertar imágenes, almacenarlas haciendo una conversión a HTTPS en la base de datos, además, tras almacenar la imagen la insertamos en el ImageView.

```
// Registrar el launcher para la cámara
cameraLauncher = registerForActivityResult(new ActivityResultContracts.StartActivityForResult(), result -> {
    if (result.getResultCode() == RESULT_OK) {
        // Si la foto se toma correctamente, mostrar la vista previa en el ImageView
        Bitmap photo = (Bitmap) result.getData().getExtras().get("data");
        // Insertar la imagen en la galería y obtenemos la URI transformada en String para almacenar en la BD
        ContentValues values = new ContentValues();
        values.put(MediaStore.Images.Media.TITLE, "personaMarfol.jpg");
        Uri uri = getContentResolver().insert(MediaStore.Images.Media.EXTERNAL_CONTENT_URI, values);
        try {
            OutputStream outputStream = getContentResolver().openOutputStream(uri);
            photo.compress(Bitmap.CompressFormat.JPEG, quality: 100, outputStream);
            outputStream.close();

            // Obtenemos la ruta URI de la imagen seleccionada
            uriCapturada = uri.toString();
            ivPlatoAnadirP.setBackground(null);
            Glide.with(activity, this) RequestManager
                .load(uriCapturada) RequestBuilder<Drawable>
                .circleCrop()
                .into(ivPlatoAnadirP);
        } catch (IOException e) {
            e.printStackTrace();
        }
        puElegirAccion.dismiss();
    }
})
```

En el caso de la galería es parecido a la hora de pedir permisos, se siguen los mismos pasos.

```
//Añadimos onClick en el ImageView para activar la imagen
btnUpGaleria.setOnClickListener(view -> {
    // Solicitar permiso para acceder a la galería
    if (ContextCompat.checkSelfPermission(context: this, Manifest.permission.READ_EXTERNAL_STORAGE) == PackageManager.PERMISSION_DENIED) {
        // Si no tenemos los permisos, los solicitamos
        ActivityCompat.requestPermissions(activity: this, new String[]{Manifest.permission.READ_EXTERNAL_STORAGE}, GALLERY_PERMISSION_CODE);
    } else {
        // Si ya se tienen los permisos, abrir la galería
        abrirGaleria();
    }
});
```

En el caso de tener los permisos accederá al método abrirGaleria.

```
if (requestCode == GALLERY_PERMISSION_CODE) {
    if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
        // Si se conceden los permisos, abrir la galería
        abrirGaleria();
    } else {
        // Si se deniegan los permisos, mostrar un mensaje al usuario
        Toast.makeText(context: this, text: "Para acceder a la galería debe otorgar los permisos", Toast.LENGTH_SHORT).show();
    }
}
```

Para acceder a la galería se debe utilizar la librería Imagepicker **DE FORMA OBLIGATORIA**.

Las distintas versiones de Android como **MIUI** bloquean el acceso a la galería y es **completamente imposible** acceder a la galería utilizando un ActivityResultLauncher, bloqueos, Excepciones, cierres inesperados.

La librería **ImagePicker** ofrece el acceso a la galería sin tener ningún tipo de problemas con versiones de Android modificadas como **MIUI**.

```
// Método para abrir la galería
2 usages ▾ Javier Calderón
private void abrirGaleria() {
    //Librería que accede a la galería del dispositivo (OBLIGATORIO USAR LIBRERÍAS MIUI BLOQUEA LO DEMÁS)
    ImagePicker.with( activity: this)
        .galleryOnly()
        .start();
}
```

Tras elegir la imagen de la galería se accede al método onActivityResult, que, tras comprobar que la URI obtenida no es null se obtiene la URI de la imagen de la galería y la insertamos en el ImageView.

```
//Método que accede a la galería sin que los permisos restrictivos MIUI afecten
+ Javier Calderón +1
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (resultCode == Activity.RESULT_OK) {
        // La Uri de la imagen no será nula para RESULT_OK
        Uri uri = data.getData();
        if (uri != null) {
            Uri selectedImageUri = data.getData();
            uriCapturada = selectedImageUri.toString();

            //Cargar imagen seleccionada
            ivPlatoAnadirP.setBackground(null);
            Glide.with( activity: this ) RequestManager
                .load(selectedImageUri) RequestBuilder<Drawable>
                .circleCrop()
                .into(ivPlatoAnadirP);
        }
        puElegirAccion.dismiss();
    }
}
```

Añadir nuevo Participante

Tras pulsar el botón “Continuar“, accedemos al método anadirComensal.

```
btnContinuarAnadirP.setOnClickListener(view -> {
    //Método encargado de crear un comensal
    anadirComensal();
});
```

Método que extrae el valor de los EditText y comprueba si el campo nombre tiene algún valor, de no tenerlo no creará el comensal y creará un mensaje de aviso.

```
public void anadirComensal() {
    boolean esValidado = true;
    nombre = String.valueOf(etNombreAnadirP.getText());
    descripcion = String.valueOf(etDescAnadirP.getText());

    if (etNombreAnadirP.getText().toString().length() == 0) {
        Toast.makeText( context: this, text: "Debe introducir un nombre", Toast.LENGTH_SHORT ).show();
        esValidado = false;
    }

    if (esValidado) {
        // Añade la persona en la base de datos
        anadirPersonaABd(nombre, descripcion, uriCapturada);

        // Añade la persona localmente
        comensales.add(new Persona(comensales.size(), nombre, descripcion, uriCapturada, platos, monedero: 0));

        intent = new Intent();
        intent.putExtra( name: "arrayListComensales", comensales );
        setResult(Activity.RESULT_OK, intent);
        finish();
    }
}
```

Como en casos anteriores tenemos un método onItemClick, siendo la posición 0 RecordarPlato (Para añadir nuevo plato) o DetallePlatoActivity, acceder al detalle de un plato ya añadido.

```
@Override
public void onItemClick(int position) {
    platoPosicion = position;
    //Si pulsas "Añadir Plato" ( 0 ), accederás a la actividad añadir plato
    if (position > 0) {
        intent = new Intent( packageContext: this, DetallePlatoActivity.class);
        intent.putExtra( name: "platoDetalle", platos.get(position));
        intent.putExtra( name: "arrayListComenComp", comensales);
        rLauncherDetallePlato.launch(intent);
    } else {
        intent = new Intent( packageContext: this, RecordarPlatoActivity.class);
        intent.putExtra( name: "arrayListPlatos", platos);
        intent.putExtra( name: "arrayListComenComp", comensales);
        intent.putExtra( name: "nombreRestaurante", nombreRestaurante);

        rLauncherRecordarPlato.launch(intent);
    }
}
```

anadirPersonaABd (LOGUEADO)

Método que tras validar el comensal creado, lo añade a la base de datos, utiliza las variables creadas en anadirComensal, filtra el INSERT utilizando el email.

Siempre que estés logueado, al añadir un comensal de forma local se añadirá automáticamente a la base de datos, de esta manera, se podrán recordar las personas en futuras cuentas

Finalmente, si el INSERT funciona correctamente, se accederá al método para subir la imagen subirlImagenPersona()

```
ate void anadirPersonaABd(String nombre, String descripcion, String imagen) {
if (currentUser != null) {
    // El usuario está autenticado
    email = currentUser.getEmail(); // Utiliza el email como ID único del usuario
    // Obtén la colección "personas" en Firestore
    personasRef = db.collection( collectionPath: "personas");
    // Realiza la consulta para verificar si ya existe una persona con los mismos valores de nombre y descripción
    consulta = personasRef.whereEqualTo( field: "nombre", nombre)
        .whereEqualTo( field: "usuarioId", email);
    consulta.get().addOnCompleteListener(task -> {
        if (task.isSuccessful()) {
            if (task.getResult().isEmpty()) {
                // No existe una persona con los mismos valores de nombre y descripción, procede a agregarla
                // Crea un objeto HashMap para almacenar los datos de la nueva persona
                Map<String, Object> nuevaPersona = new HashMap<>();
                nuevaPersona.put( k: "nombre", nombre);
                nuevaPersona.put( k: "imagen", v: "");
                nuevaPersona.put( k: "descripcion", descripcion);
                nuevaPersona.put( k: "usuarioId", email);
                // Agrega la nueva persona con un ID único generado automáticamente
                personasRef.add(nuevaPersona)
                    .addOnSuccessListener(documentReference -> {
                        // La persona se agregó exitosamente
                        personaId = documentReference.getId();
                        subirlImagenPersona(personaId, imagen);
                    })
                    .addOnFailureListener(e -> {
                    });
            }
        }
    });
}
```

subirImagenPersona

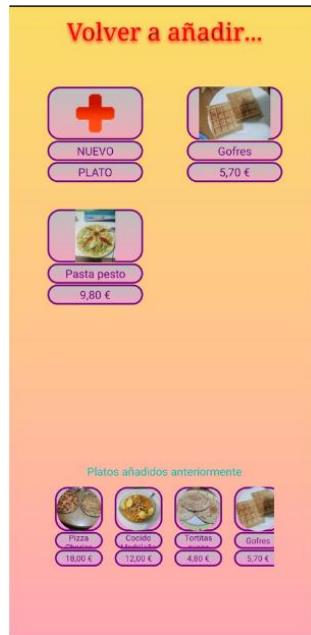
Se debe extraer la referencia de la colección, obtenemos la ruta y finalmente comprobamos que existe una imagen.

De esta manera podemos insertar la imagen a la persona correcta.

```
private void subirImagenPersona(String personaId, String imagen) {
    if (personaId != null && !personaId.isEmpty()) {
        // Obtiene una referencia al Storage de Firebase
        storageRef = FirebaseStorage.getInstance().getReference();
        // Define una referencia a la imagen en Storage utilizando el ID de la persona
        rutaImagen = "personas/" + personaId + ".jpg";
        imagenRef = storageRef.child(rutaImagen);
        // Sube la imagen a Storage
        if (!imagen.equalsIgnoreCase("")){
            Uri imagenUri = Uri.parse(imagen);
            uploadTask = imagenRef.putFile(imagenUri);
            uploadTask.addOnSuccessListener(taskSnapshot -> {
                // Obtiene la URL de descarga de la imagen
                imagenRef.getDownloadUrl().addOnSuccessListener(uri -> {
                    // La URL de descarga de la imagen está disponible
                    imageUrl = uri.toString();
                    // Actualiza el campo "imagen" en Firestore con la URL de descarga de la imagen
                    db.collection( collectionPath: "personas").document(personaId).DocumentReference
                        .update( field: "imagen", imageUrl) Task<Void>
                        .addOnSuccessListener(aVoid -> {
                            // La URL de la imagen se actualizó exitosamente en Firestore
                        })
                        .addOnFailureListener(e -> {
                            // Ocurrió un error al actualizar el campo "imagen" en Firestore
                        });
                });
            }).addOnFailureListener(e -> {
        });
    }
}
```

Recordar platos

Actividad que inicializa al pulsar (+) para añadir un plato nuevo, esta presenta formatos parecidos a los anteriores.



- Primera lista: Muestra el botón 0 para añadir nuevo plato y el resto se tratan de platos que has añadido en esa cuenta en concreto es decir, solo mostrará los platos que has añadido en ese momento, agilizando bastante la gestión de alimentos, normalmente, varias personas repiten una bebida por ejemplo.
- Segunda lista (LOGUEADO) *: Mostrará todos los platos que hayas añadido anteriormente en ese restaurante en concreto. Es importante resaltar que cada restaurante puede tener productos iguales con distinto precio, etc.

En el caso que el usuario desee añadir un plato completamente nuevo accederá a AnadirPlatoActivity al pulsar el botón de la posición 0 “ NUEVO PLATO ”



Si el usuario pulsa un plato añadido anteriormente en esa cuenta lo añadirá directamente a la lista de platos sin realizar más comprobaciones



(LOGUEADO) Parecido a casos anteriores la lista inferior muestra elementos de la base de datos pertenecientes a ese restaurante, siempre que estés logueado, al pulsar sobre ellos se añade directamente sin realizar más comprobaciones



RecordarPlatoActivity cuenta con los métodos generales para asignar efectos, activar la lista, etc.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_recordar_plato);
    intent = getIntent();
    nombreRestaurante = intent.getStringExtra("nombreRestaurante");

    //Recibe la lista de platos y filtra los repetidos
    recibirDatos();

    //Método que asigna IDs a los elementos
    asignarId();

    //Método que asigna los efectos a los elementos
    asignarEfectos();

    //Método que muestra el contenido del adaptador
    mostrarAdapter();

    //Muestra adapter bd
    mostrarAdapterBd();
```

onItemClick

Similar a métodos añadidos anteriormente, el elemento de la posición 0 se encarga de enviar a la actividad padre AnadirParticipanteActivity un boolean indicando que es un plato nuevo y no un plato recordado.

De ser un plato recordado esta clase devuelve a la clase padre el plato seleccionado.

```

@Override
public void onItemClick(int position) {
    if (position < 1) {
        //Se va a añadir un plato nuevo accedemos a su actividad
        intent = new Intent();
        intent.putExtra( name: "nuevoPlato", value: true);
        setResult(Activity.RESULT_OK, intent);
        finish();
    } else {
        //Pasamos el plato a no compartido
        listaPlatos.get(position).setCompartido(false);
        intent = new Intent();
        intent.putExtra( name: "recordarNuevo", listaPlatos.get(position));
        setResult(Activity.RESULT_OK, intent);
        finish();
    }
}

```

En la lista de platos recordados (**LOGUEADO**)

Esta realiza la misma acción que el método anterior, reenvía al padre el plato recordado

```

@Override
public void onItemClickBd(int position) {
    intent = new Intent();
    plato = new Plato(listaPlatosBd.get(position).getNombre(),listaPlatosBd.get(position).getDescripcion(),listaPlatosBd.get(position));
    intent.putExtra( name: "recordarNuevo", plato);
    setResult(Activity.RESULT_OK, intent);
    finish();
}

```

Añadir plato

Actividad en la cual se crea un **plato nuevo**, de temática parecida a AnadirParticipanteActivity, cuenta con distintos EditText, ImageView con acceso a cámara y galería, etc.

La actividad obliga a insertar un **nombre** y un **precio**, además, cuenta con una de nuestras mejores y más complejas funcionalidades: **Compartir**; funcionalidad que permite repartir el valor de un plato entre los comensales insertados, de esta manera, puedes repartir un entrante entre aquellos que hayan participado.

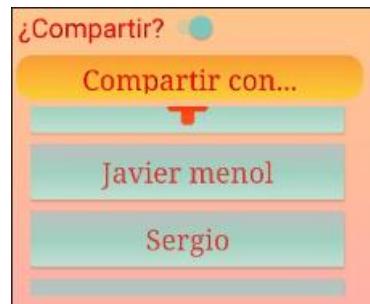


Switch ¿Compartir?:

Switch que habilita la lista de Compartir, en ella añades las personas con quien quieras compartir el plato en concreto



Al ser una lista puedes comprobar aquellas personas que has añadido, o incluso, añadir más.



AnadirPlatoActivity cuenta con los métodos generales para estilos, adaptadores, etc.

```
//Recibe la lista de comensales para empezar a añadir
intent = getIntent();
platos = (ArrayList<Plato>) intent.getSerializableExtra( name: "arrayListPlatos");

//Recibo desde Editar o Añadir
nombreCompartir = (ArrayList<Persona>) intent.getSerializableExtra( name: "arrayListComenComp");
personaCode = intent.getIntExtra( name: "comensalCode", defaultValue: 0);

//Método que asigna IDs a los elementos
asignarId();

//Método que asigna los efectos a los elementos
asignarEfectos();

//Método que muestra el contenido del adaptador
mostrarAdapter();
```

El switch posee el método setOnCheckedChangeListener que es capaz de detectar cuando un switch está activado o no, en el caso que esté activado este habilitará la lista de compartidos.

```
//Comprueba si el switch compartir está activo o no para mostrar su información
swCompartirPlato.setOnCheckedChangeListener((compoundButton, isChecked) -> {
    if (isChecked) {
        tvSubTitP.setVisibility(View.VISIBLE);
        rvPlatosAnadirPlato.setVisibility(View.VISIBLE);
        esCompartido=true;
    } else {
        tvSubTitP.setVisibility(View.INVISIBLE);
        rvPlatosAnadirPlato.setVisibility(View.INVISIBLE);
        esCompartido=false;
    }
});
```

Esta clase repite los métodos de cámara y galería a la hora de añadir una imagen

```
//Añadimos onClick en el ImageView para activar la imagen
btnUpCamara.setOnClickListener(view -> {
    // Solicitar permiso para acceder a la cámara
    if (ContextCompat.checkSelfPermission(context: this, Manifest.permission.CAMERA) == PackageManager.PERMISSION_DENIED) {
        //Si no tenemos los permisos los obtenemos
        ActivityCompat.requestPermissions(activity: this, new String[]{Manifest.permission.CAMERA}, CAMERA_PERMISSION_CODE);
    } else {
        // Si ya se tienen los permisos, abrir la cámara
        abrirCamara();
    }
});

//Añadimos onClick en el ImageView para activar la imagen
btnUpGaleria.setOnClickListener(view -> {
    // Solicitar permiso para acceder a la galería
    if (ContextCompat.checkSelfPermission(context: this, Manifest.permission.READ_EXTERNAL_STORAGE) == PackageManager.PERMISSION_DENIED) {
        // Si no tenemos los permisos, los solicitamos
        ActivityCompat.requestPermissions(activity: this, new String[]{Manifest.permission.READ_EXTERNAL_STORAGE}, GALLERY_PERMISSION_CODE);
    } else {
        // Si ya se tienen los permisos, abrir la galería
        abrirGaleria();
    }
});
```

anadirPlato

El botón Continuar para finalizar el plato, activa el método anadirPlato() que valida los contenidos añadidos.

```
//Botón encargado de añadir el plato
btnContinuarAnadirP.setOnClickListener(view -> { anadirPlato(); });
```

Antes de crear el plato, parecido a anadirParticipante, se validan los EditText para ver si tienen contenido, en el caso del precio a parte de validar que posea un valor, este remplazará los caracteres “ , ” por “ . ” ya que los valores números se gestionan utilizando puntos.

```
public void anadirPlato () {
    boolean esValidado=true;
    nombre = String.valueOf(etNombreAnadirP.getText());
    descripcion = String.valueOf(etDescAnadirP.getText());
    //Comprueba si el editText está vacío, de estarlo el programa lo entiende como un 0, además, remplaza <,> por <,> para evitar errores
    precio = etPrecioAnadirP.getText().toString().equalsIgnoreCase("anotherString: """) ? 0 : Double.parseDouble(etPrecioAnadirP.getText().toString().replace(",","."));
    //Comprueba si has añadido un nombre
    if (etNombreAnadirP.getText().toString().length() == 0) {
        Toast.makeText(context: this, text: "Debe introducir un nombre", Toast.LENGTH_SHORT).show();
        esValidado=false;
    }
    if (precio <= 0) {
        Toast.makeText(context: this, text: "Debe introducir un precio mayor a 0", Toast.LENGTH_SHORT).show();
        esValidado=false;
    }
}
```

Continuando el método, tras comprobar que es validado, se comprueba si es un plato **compartido**, en el caso de que lo sea, primero se borra la posición 0 de la lista ya que es el botón (“**NUEVO PLATO**”), acto seguido, se crea el objeto Plato con todos sus valores y su lista de compartidos, que, se añade a la lista de platos que posee ese comensal en concreto.

En caso que la lista esté activada pero no hayan comensales compartidos, se considerará un plato no compartido.

Por el lado contrario, si no es compartido, creará el plato con una lista vacía pues no se ha compartido con nadie.

```
//Comprueba si ha validado, añade la persona, envía al padre el ArrayList de comensales, platos y cierra la actividad
if (esValidado) {
    //Dependiendo si es compartido o no, sigue un
    if (esCompartido) {
        //Eliminamos la posición 0 ya que es un botón
        comensalCompartirList.remove( index: 0 );
        //Hacemos una última comprobación si ha añadido personas en compartir, si no ha añadido se añade un plato NO compartido
        if (comensalCompartirList.size()>0) {
            //Añado a la lista la persona creada
            platos.add(new Plato(nombre,descripcion,precio,precio,uriCapturada,esCompartido, comensalCompartirList));
        } else {
            //Añado a la lista la persona creada
            esCompartido=false;
            platos.add(new Plato(nombre,descripcion,precio,precio,uriCapturada,esCompartido, new ArrayList<>()));
        }
    } else {
        //Añado a la lista la persona creada
        platos.add(new Plato(nombre,descripcion,precio,precio,uriCapturada,esCompartido, new ArrayList<>()));
    }
    intent = new Intent();
    intent.putExtra( name: "arrayListPlatos", platos);
    setResult(Activity.RESULT_OK, intent);
    finish();
}
```

onItemClick

Método extremadamente importante, similar a métodos anteriores, la posición 0 se encarga de abrir la actividad CompartirListaActivity, esta recibe un ArrayList con los participantes añadidos en la cuenta, el que selecciones compartirá el plato y repartirá el precio a partes iguales, sin tener un límite de personas.

Para que CompartirListaActivity pueda presentar una lista de comensales sin repetir a aquellos que hayas añadido, esta actividad envía una lista filtrada, es decir, si el plato ha sido compartido con “Juan” la lista enviada no podrá tener a “Juan” ya que ya está añadido.

Para poder filtrar la lista de forma óptima, utilizamos un método llamado `.removeIf()` que requiere minSDK 24.

Como comprobaremos a continuación, la lista de personas compartidas filtrada se trata de “comensalCompartirList” y la lista de todos los comensales sería nombreCompartir

Se utiliza una nueva lista llamada noRepCompartirList, que es aquella que borra los repetidos

```
@Override
public void onItemClick(int position) {
    if (position==0) {
        //Método que comprueba si hay repetidos en la lista de compartidos (no aparezcan de nuevo)
        //Además, comprueba que no esté el propio usuario a la hora de repartir
        noRepCompartirList = nombreCompartir;

        for (Persona p : comensalCompartirList) {
            noRepCompartirList.removeIf(persona -> persona.getComensalCode() == p.getComensalCode());
            noRepCompartirList.removeIf(persona -> persona.getComensalCode() == personaCode);
        }

        //Accedemos a la actividad de compartir plato
        intent = new Intent(getApplicationContext(), CompartirListaActivity.class);
        intent.putExtra("name: "arrayListComenComp", noRepCompartirList);
        rLauncherComp.launch(intent);
    }
}
```

ActivityResultLauncher que recibe la persona con quien se desea compartir, de esta manera, finalmente, podremos añadir en cada plato todas las personas que lo compartirán de forma precisa y exacta.

```
//Launcher Result recibe las personas de la lista que van a compartir
rLauncherComp = registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(), result -> {
    if (result.getResultCode() == Activity.RESULT_OK) {
        //Método de cálculo aquí también
        Intent data = result.getData();
        comensalCompartirList.add((Persona) data.getSerializableExtra("personaCompartir"));
        anadirPAdapter.setResultsPersonaCom(comensalCompartirList);
    }
});
```

Compartir lista

Actividad muy sencilla pero vital, como vimos anteriormente, esta actividad ofrece los comensales que has añadido anteriormente, para que puedas elegir a aquellos con quienes quieres compartir el plato.

En esta actividad no existe un botón para añadir ya que se espera elegir a un comensal para compartir, de pulsar uno en concreto, la actividad devolverá al Padre el comensal pulsado y esta se cerrará con un finish.



Esta actividad cuenta con los métodos generales para asignar colores, efectos, etc.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_compartir_lista);

    Intent intentObtener = getIntent();
    nombreCompartir = (ArrayList<Persona>) intentObtener.getSerializableExtra("arrayListComenComp");

    //Método que asigna IDs a los elementos
    asignarId();

    //Método que asigna los efectos a los elementos
    asignarEfectos();

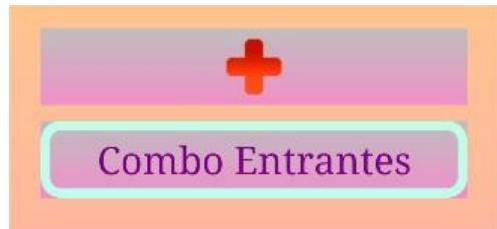
    //Método que muestra el contenido del adaptador
    mostrarAdapter();
}
```

onItemClick

En el caso de elegir un comensal para compartir, este es seleccionado y enviado al padre, acto seguido, se cierra la actividad

```
• Javier Calderón
@Override
public void onItemClick(int position) {
    //Devuelve la persona que va a compartir
    Intent intentPersonaCom = new Intent();
    intentPersonaCom.putExtra( name: "personaCompartir", nombreCompartir.get(position));
    setResult(Activity.RESULT_OK, intentPersonaCom);
    finish();
}
```

Tras añadir el plato, si es compartido, en la lista de platos se vería con un borde celeste indicando que es compartido.



Si pulsas un plato ya añadido como el siguiente se accederá a DetallePlatoActivity.



Actividad que da opción a modificar un plato, consideramos que, un usuario podría a llegar equivocarse, o haber añadido algo mal, siempre se tiene la opción a modificar todo.

Podemos comprobar que, tenemos imágenes por defecto para aquellos elementos que no tienen imagen, en el caso de los platos, es el icono Marfol morado, siguiendo la temática de colores.

El plato que añadimos anteriormente, era un plato compartido, como se puede comprobar, aquí volvemos a tener la lista y en este caso está activada, de ser un plato no compartido saldría desactivada.

Se puede tanto editar como borrar un plato, similar a DetallePersonaActivity

Similar a la actividad de AñadirPlatoActivity, podemos gestionar todos los atributos que crean un plato, en el caso de no tener imagen, se añade la imagen de marfol, con su respectivo color



Si el plato que has pulsado era compartido tanto el switch como la lista estaría activados pues se trata de un plato compartido, desde este punto también se puede gestionar por posibles errores.



Finalmente, los botones editar y borrar, editar modifica los valores, como hemos visto anteriormente en AnadirParticipanteActivity, posee un ActivityResultLauncher que dependiendo que botón elijas aquí, hará una función u otra



Esta actividad cuenta con los métodos generales para asignar colores, efectos, etc.

```
//Recibe la lista de comensales para empezar a añadir
intent = getIntent();
plato = (Plato) intent.getSerializableExtra( name: "platoDetalle");
nombreCompartir = (ArrayList<Persona>) intent.getSerializableExtra( name: "arrayListComenComp");
personaCode = intent.getIntExtra( name: "comensalCode", defaultValue: 0);
esCompartido = plato.isCompartido();

//Método que asigna IDs a los elementos
asignarId();

//Método que asigna los efectos a los elementos
asignarEfectos();

//Método que muestra el contenido del adaptador
mostrarAdapter();

//Método que inserta la información a los comensales
insertarComensal();
```

Similar a AnadirPlatoActivity el switch tiene el mismo funcionamiento, de estar activado muestra los elementos orientados a compartir, de no estarlo, funcionamiento normal.

```
//Comprueba si el switch compartir está activo o no para mostrar su información
swCompartirPlato.setOnCheckedChangeListener((compoundButton, isChecked) -> {
    if (isChecked) {
        tvListaEditarPlato.setVisibility(View.VISIBLE);
        rvAnadirPlatoDetalle.setVisibility(View.VISIBLE);
        esCompartido = true;
    } else {
        tvListaEditarPlato.setVisibility(View.INVISIBLE);
        rvAnadirPlatoDetalle.setVisibility(View.INVISIBLE);
        esCompartido = false;
    }
});
```

Repite el mismo funcionamiento que AnadirPlatoActivity, de hecho, reutiliza la misma actividad, para ambos caso se abre la misma actividad CompartirListaActivity

```
//Laucher Result recibe las personas de la listas que van a compartir
rLauncherComp = registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(), result -> {
        if (result.getResultCode() == Activity.RESULT_OK) {
            //Método de cálculo aquí también
            intent = result.getData();
            plato.getPersonasCompartir().add((Persona) intent.getSerializableExtra("name: \"personaCompartir\""));
            anadirPAdapter.setResultsPersonaCom(plato.getPersonasCompartir());
        }
    }
);
```

Esta clase repite los métodos de cámara y galería a la hora de añadir una imagen

```
//Añadimos onClick en el ImageView para activar la imagen
btnUpCamara.setOnClickListener(view -> {
    // Solicitar permiso para acceder a la cámara
    if (ContextCompat.checkSelfPermission(context: this, Manifest.permission.CAMERA) == PackageManager.PERMISSION_DENIED) {
        //Si no tenemos los permisos los obtenemos
        ActivityCompat.requestPermissions(activity: this, new String[]{Manifest.permission.CAMERA}, CAMERA_PERMISSION_CODE);
    } else {
        // Si ya se tienen los permisos, abrir la cámara
        abrirCamara();
    }
});
//Añadimos onClick en el ImageView para activar la imagen
btnUpGaleria.setOnClickListener(view -> {
    // Solicitar permiso para acceder a la galeria
    if (ContextCompat.checkSelfPermission(context: this, Manifest.permission.READ_EXTERNAL_STORAGE) == PackageManager.PERMISSION_DENIED) {
        // Si no tenemos los permisos, los solicitamos
        ActivityCompat.requestPermissions(activity: this, new String[]{Manifest.permission.READ_EXTERNAL_STORAGE}, GALLERY_PERMISSION_CODE);
    } else {
        // Si ya se tienen los permisos, abrir la galeria
        abrirGaleria();
    }
});
```

Pulsar el botón borrar activa el método borrarPlato()

```
//Botón de borrar, envía a la actividad padre un boolean indicando si debe borrar
btnBorrarDetalle.setOnClickListener(view -> {
    borrarPlato();
});
```

Método sencillo que, envía el boolean al Launcher de la actividad padre, indicando que se debe borrar el plato.

```
public void borrarPlato() {
    Intent borrarPlato = new Intent();
    borrarPlato.putExtra( name: "borrarPlato", value: true);
    setResult(Activity.RESULT_OK, borrarPlato);
    finish();
}
```

Editar desemboca en el método editarPlato()

```
//Botón de editar, envía un nuevo plato tras ser validado y sustituye el antiguo
btnEditarDetalle.setOnClickListener(view -> {
    editarPlato();
});
```

Método muy similar a anadirPlato, realiza las mismas comprobaciones, cambia “ , ” por “ . ”, etc.

```
public void editarPlato() {
    boolean esValidado = true;
    nombre = String.valueOf(etTitleDetalle.getText());
    descripcion = String.valueOf(etDescripcionDetalle.getText());
    //Comprueba si el editText está vacío, de estarlo el programa lo entiende como un 0, además, remplaza <,> por <.
    precio = etPrecioDetalle.getText().toString().equalsIgnoreCase( anotherString: "" ) ? 0 : Double.parseDouble(etPrecioD
    //Comprueba si has añadido un nombre
    if (etTitleDetalle.getText().toString().length() == 0) {
        Toast.makeText( context: this, text: "Debe introducir un nombre", Toast.LENGTH_SHORT).show();
        esValidado = false;
    }
    if (precio <= 0) {
        Toast.makeText( context: this, text: "Debe introducir un precio mayor a 0", Toast.LENGTH_SHORT).show();
        esValidado = false;
    }
}
```

Realiza las mismas comprobaciones, si es validado, si es compartido, en el caso de que esté todo validado, se reenviará al padre el nuevo plato editado, en AnadirParticipante, como se ha visto anteriormente, se cambiará el plato antiguo por el nuevo modificado.

```
//Comprueba si ha validado, añade la persona, envia al padre el ArrayList de comensales, platos y cierra la actividad
if (esValidado) {
    //Dependiendo si es compartido o no
    if (esCompartido) {
        //Eliminamos la posición 0 ya que es un botón
        plato.getPersonasCompartir().remove( index: 0 );

        //Hacemos una Última comprobación si ha añadido personas en compartir, si no ha añadido se añade un plato NO compartido
        if (plato.getPersonasCompartir().size() > 0) {
            //Añado a la lista la persona creada
            - lista de personas que van a compartir el plato
            platoEditado = new Plato(nombre, descripcion, precio, precio, uriCapturada, esCompartido, plato.getPersonasCompartir());
        } else {
            //Añado a la lista la persona creada
            esCompartido = false;
            platoEditado = new Plato(nombre, descripcion, precio, precio, uriCapturada, esCompartido, new ArrayList<>());
        }
    } else {
        //Añado a la lista la persona creada
        //Solo se crea el plato si en el compartido
        platoEditado = new Plato(nombre, descripcion, precio, precio, uriCapturada, esCompartido, new ArrayList<>());
    }
}

Intent intentComensal = new Intent();
intentComensal.putExtra( name: "detallePlato", platoEditado );
setResult(Activity.RESULT_OK, intentComensal);
finish();
```

insertarComensal

Ya que la actividad es EditarPlatoActivity los datos del plato a editar se han de insertar automáticamente al abrir la actividad.

```
//Método que inserta la información a los comensales
insertarComensal();
```

Se insertan en las vistas los datos del plato a editar, en el caso de tener una imagen, se insertará.

Además, se aplica un cambio de color a la progressBar, se utiliza el color rojo característico de la aplicación

```
public void insertarComensal() {
    etTitleDetalle.setText(plato.getNombre());
    etDescripcionDetalle.setText(plato.getDescripcion());
    etPrecioDetalle.setText(String.valueOf(plato.getPrecio()));
    uriCapturada = plato.getUrlImage();
    //Comprobamos que el plato no es nulo
    if (plato.getUrlImage() != null && !plato.getUrlImage().equalsIgnoreCase( anotherString: "" )) {
        //Asignamos un color rojizo característico de la APP
        progressBar.getIndeterminateDrawable().setColorFilter(
            ContextCompat.getColor( context this, R.color.redSLight),
            PorterDuff.Mode.SRC_IN
        );
    }
}
```

En el caso de tener una imagen, se inserta la imagen junto al método .listener que da la opción a mejorar la experiencia considerablemente.

```
Glide.with( activity: this) RequestManager
    .load(plato.getUrlImage()) RequestBuilder<Drawable>
    .diskCacheStrategy(DiskCacheStrategy.NONE)
    .circleCrop()
    .error(uriCapturada)
    .listener(new RequestListener<Drawable>() {
        @ Javier Calderón
        @Override
        public boolean onLoadFailed(@Nullable GlideException
            progressBar.setVisibility(View.GONE);
            return false;
        }

        @ Javier Calderón
        @Override
        public boolean onResourceReady(Drawable resource, Object
            progressBar.setVisibility(View.GONE);
            return false;
        }
    })
    .into(ivFotoDetalle);
```

onItemClick de EditarPlatoActivity, similar a AnadirPlatoActivity, se comparan ambos ArrayList de comensales, eliminando los repetidos en la lista de compartidos.

```
@ Javier Calderón +1
@Override
public void onItemClick(int position) {
    if (position == 0) {
        //Método que comprueba si hay repetidos en la lista de compartidos (no aparezcan de nuevo)
        //Además, comprueba que no esté el propio usuario a la hora de repartir
        noRepComList = nombreCompartir;

        for (Persona p : plato.getPersonasCompartir()) {
            noRepComList.removeIf(persona -> persona.getComensalCode() == p.getComensalCode());
            noRepComList.removeIf(persona -> persona.getComensalCode() == personaCode);
        }

        //Accedemos a la actividad de compartir plato
        intent = new Intent( packageContext: this, CompartirListaActivity.class);
        intent.putExtra( name: "arrayListComenComp", noRepComList);
        rLauncherComp.launch(intent);

    }
}
```

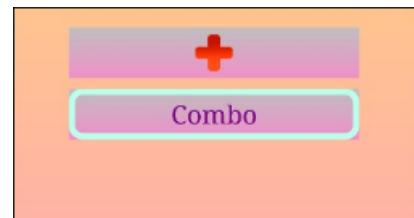
Detalle persona

Actividad detalle de un comensal, así como puedes editar un plato, puede editar un comensal completamente, cambiar su nombre, su descripción, sus platos, también se puede borrar el comensal por si hay un cambio en la cuenta.

Se debe aclarar que, no modificarán nada en la base de datos, para modificar valores en la base de datos, en el menú se tiene acceso a Editar desde ahí gestionas todos los elementos de la base de datos



Desde EditarPlatoActivity podemos comprobar los platos añadidos anteriormente, incluso podemos identificarlos por su nombre o en el ejemplo, un plato compartido.



Similar a EditarPlatoActivity, inserta en la actividad los valores de la persona pulsada, utiliza para la imagen Glide con el método .listener.

```
public void insertarComensal() {
    etTitleDetalle.setText(comensal.getNombre());
    etDescripcionDetalle.setText(comensal.getDescripcion());
    uriCapturada = comensal.getUrlImage();
    if (comensal.getUrlImage() != null && !comensal.getUrlImage().equalsIgnoreCase("anotherString: """)) {
        //Asignamos un color rojizo característico de la APP
        progressBar.getIndeterminateDrawable().setColorFilter(
            ContextCompat.getColor(context: this, R.color.redSLight),
            PorterDuff.Mode.SRC_IN
    );
}
```

Desde DetallePersonaActivity se reutilizan actividades para añadir plato como AnadirPlatoActivity.

La posición 0 simula ser un botón como hemos visto hasta ahora y mayores se accede a DetallePlatoActivity.

```
▲ Javier Calderón +3
@Override
public void onItemClick(int position) {
    platoPosicion = position;
    //Si pulsas "Añadir Persona" ( 0 ), accederás a la actividad añadir persona
    if (position > 0) {
        Intent intentDetalle = new Intent( packageContext: this, DetallePlatoActivity.class);
        intentDetalle.putExtra( name: "platoDetalle", platos.get(position));
        intentDetalle.putExtra( name: "arrayListComenComp", nombreCompartir);
        intentDetalle.putExtra( name: "comensalCode", comensal.getComensalCode());
        rLauncherDetallePlato.launch(intentDetalle);
    } else {
        Intent intent = new Intent( packageContext: this, RecordarPlatoActivity.class);
        intent.putExtra( name: "arrayListPlatos", platos);
        intent.putExtra( name: "arrayListComenComp", nombreCompartir);
        intent.putExtra( name: "nombreRestaurante", nombreRestaurante);
        intent.putExtra( name: "enviarRestaurante", value: true);
        rLauncherRecordarPlato.launch(intent);
    }
}
```

Similar a las clases anteriores, el funcionamiento de la cámara y la galería, es el mismo.

```
//Añadimos onClick en el ImageView para activar la imagen
btnUpCamara.setOnClickListener(view -> {
    // Solicitar permiso para acceder a la cámara
    if (ContextCompat.checkSelfPermission(context, Manifest.permission.CAMERA) == PackageManager.PERMISSION_DENIED) {
        //Si no tenemos los permisos los obtenemos
        ActivityCompat.requestPermissions(activity, new String[]{Manifest.permission.CAMERA}, CAMERA_PERMISSION_CODE);
    } else {
        // Si ya se tienen los permisos, abrir la cámara
        abrirCamara();
    }
});

//Añadimos onClick en el ImageView para activar la imagen
btnUpGaleria.setOnClickListener(view -> {
    // Solicitar permiso para acceder a la galería
    if (ContextCompat.checkSelfPermission(context, Manifest.permission.READ_EXTERNAL_STORAGE) == PackageManager.PERMISSION_DENIED) {
        // Si no tenemos los permisos, los solicitamos
        ActivityCompat.requestPermissions(activity, new String[]{Manifest.permission.READ_EXTERNAL_STORAGE}, GALLERY_PERMISSION_CODE);
    } else {
        // Si ya se tienen los permisos, abrir la galería
        abrirGaleria();
    }
});
```

Desglose

Actividad final que muestra en una lista el reparto justo y equitativo de lo que tiene que pagar cada usuario en la cuenta.

Se debe destacar que, todos los calculos, reparto de platos, reparto de platos compartidos se realizan desde esta actividad, favoreciendo que en actividades anteriores, el usuario tenga el control total de modificar todo, es decir, puede volver cambiar todo y al ir al Desglose, volver a tener todo actualizado y repartido



Guardar (LOGUEADO)

Si no estás logueado, este botón te enviará directamente a la actividad de AuthActivity para que el usuario sepa que es necesario loguearse.



Si estás logueado, aparecerá un pop-up que te pedirá el nombre del restaurante en el caso de ser una cuenta nueva, podrás guardar el restaurante y todos los platos que añadiste en esa cuenta serán recordado siempre que lo vuelvas a utilizar



En el caso de que sea un restaurante importado a parte de poder recordar sus platos, a la hora de guardar el restaurante, no podrás modificar el nombre, aparecerá el nombre de ese restaurante importado.



En este caso MarTest, sin embargo, es muy importante aclarar que, si seleccionas guardar todos los nuevos platos y aquellos antiguos editados serán modificados y/o añadidos al restaurante.

Se puede dar en aquellos restaurantes que pueden subir o bajar el precio de un plato, de esta manera, siempre tendremos los datos actualizados



Enviar el desglose a Whatsapp

Una de las funcionalidades más útiles de Marfol es el reparto en Whatsapp, esta enviará el reparto justo en un formato texto para que el usuario pueda mostrar de forma mucho más fácil y rápida el reparto, por ejemplo, en un grupo de whatsapp, agilizando aún más el proceso.



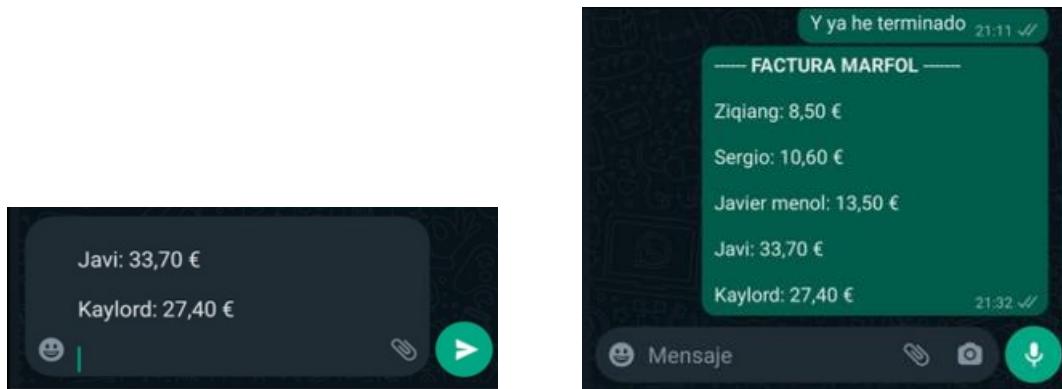
Desglose realizado y cuenta repartida, es una lista como las mostradas anteriormente.

Se debe pulsar el icono de Whatsapp

Al elegir un chat, aparecerá en el propio bocadillo de texto, toda la cuenta en formato de texto.

Se ha añadido un pequeño encabezado indicando el inicio de la cuenta

Esta ha demostrado ser una de las funcionalidades más útiles de Marfol.



A parte de contar con los métodos generales para los efectos, hace una comprobación si es una cuenta de un restaurante importado o no, de serlo, como se ha comentado anteriormente, se desactiva el EditText.

```
//Recibe la lista de comensales para empezar a añadir
intent = getIntent();
comensales = (ArrayList<Persona>) intent.getSerializableExtra( name: "envioDesglose");
nombreRestaurante = intent.getStringExtra( name: "nombreRestaurante");
//Método que asigna IDs a los elementos
asignarId();
if (nombreRestaurante != null) {
    etNombrePopup.setText(nombreRestaurante);
    etNombrePopup.setFocusable(false);
    Toast.makeText( context: this, nombreRestaurante, Toast.LENGTH_SHORT).show();
}

//Método que asigna los efectos a los elementos
asignarEfectos();

//Comprobar si el usuario está logueado
comprobarLauncher();

//Reparte los platos compartidos entre los comensales
repartirCompartido();

//Método que muestra el contenido del adaptador
mostrarAdapter();
```

(LOGUEADO) De estar logueado cuando pulsas guardar, activa el pop-up para guardar Restaurante.

```
//Botón para guardar restaurante, añadimos el texto deseado al pop up
btnGuardarRestaurante.setOnClickListener(view -> {
    //comprueba si el usuario está logueado
    if (currentUser != null) {
        //si el usuario está logueado te muestra el pop up por si le interesa guardar el restaurante
        tvMessage1Popup.setText("¿ Deseas guardar el restaurante ?");
        tvMessage2Popup.setText("Podrás recordar los platos muy fácilmente");
        puGuardarDesglose.show();
        //Método cancelar para no guardar en la bd
        btnNoGuardarDesglose.setOnClickListener(v1 -> {
            Toast.makeText(context: this, text: "Gracias por utilizar marfol", Toast.LENGTH_SHORT).show();
            puGuardarDesglose.dismiss();
            finish();
        });
    }
});
```

De no estar logueado, Marfol invita al usuario a registrarse.

```
//si no estoy logueado no saldrá ningún pop up y se finalizará la actividad
Toast.makeText(context: this, text: "Regístrate para guardar el restaurante", Toast.LENGTH_SHORT).show();
intent = new Intent(packageContext: this, Login.AuthActivity.class);
rLauncherLogin.launch(intent);
```

Si estás logueado y pulsas GUARDAR, se accede al método guardarRestaurante encargado de guardar en la base de datos el restaurante creado.

```
//Guarda el restaurante en la BD :)
btnConfirmarDesglose.setOnClickListener(v2 -> {
    nombreRestaurante = String.valueOf(etNombrePopup.getText());
    email = currentUser.getEmail();
    //compruebo si el nombre del restaurante está vacío
    if (!(nombreRestaurante.equalsIgnoreCase(anotherString: ""))) {
        copiarPlatosEnNuevoArray();
        guardarRestaurante(nombreRestaurante, email);
        Toast.makeText(context: this, text: "Se han guardado los platos", Toast.LENGTH_SHORT).show();
        puGuardarDesglose.dismiss();
        finish();
    }
});
```

Parecido a métodos anteriores, extraemos el email del usuario para crear las nuevas operaciones en la base de datos, de ser un restaurante nuevo lo crea, de existir lo actualiza.

```

private void guardarRestaurante(String nombreRestaurante, String correoUsuario) {
    // Consultar si ya existe un restaurante con el mismo nombre y usuario
    query = db.collection( collectionPath: "restaurantes").whereEqualTo( field: "nombreRestaurante", nombreRestaurante).whereEqualTo( field: "correoUsuario", correoUsuario)
    query.get().addOnCompleteListener(task -> {
        if (task.isSuccessful()) {
            querySnapshot = task.getResult();
            if (querySnapshot.isEmpty()) {
                // No existe un restaurante con el mismo nombre y usuario, se guarda uno nuevo
                // Crear un nuevo documento en la colección "restaurantes"
                restauranteRef = db.collection( collectionPath: "restaurantes").document();
                // Crear un mapa con los datos del restaurante
                Map<String, Object> restauranteData = new HashMap<>();
                restauranteData.put( k: "nombreRestaurante", nombreRestaurante);
                restauranteData.put( k: "usuarioId", correoUsuario);
                // Guardar el restaurante en la base de datos
                restauranteRef.set(restauranteData).addOnSuccessListener(aVoid -> {
                    Toast.makeText( context: this, text: "Restaurante guardado correctamente", Toast.LENGTH_SHORT).show();
                    guardarPlatos(nombreRestaurante); // Llamar al método para guardar los platos
                    guardarHistorial(nombreRestaurante, email); // Llamar al método para guardar un historial de la transacción
                    puGuardarDesglose.dismiss();
                    finish();
                }).addOnFailureListener(e -> {
                    puGuardarDesglose.dismiss();
                    finish();
                });
            }
        }
    });
}

```

Tras guardarRestaurante se activan los siguientes métodos, guardarHistorial y guardarPlatos.

```

guardarHistorial(nombreRestaurante, email); // Llamar al método para guardar un historial de la transacción
guardarPlatos(nombreRestaurante); // Llamar al método para guardar los platos
puGuardarDesglose.dismiss();
finish();

```

guardarHistorial

Utilizando el método cargarHistorial() transformamos los objetos en una cadena que posee todas las personas y precios separados por saltos de línea, acto seguido se guarda en la base de datos

```

private void guardarHistorial(String nombreRestaurante, String email) {
    if (currentUser != null) {
        // Obtén la colección "historial" en Firestore
        historialRef = db.collection( collectionPath: "historial");
        cargarHistorial();
        // Crea un objeto HashMap para almacenar los datos del nuevo historial
        Map<String, Object> nuevoHistorial = new HashMap<>();
        nuevoHistorial.put( k: "historial", historial);
        nuevoHistorial.put( k: "restaurante", nombreRestaurante);
        // Obtiene la fecha actual en el formato deseado ("dd/MM/yyyy HH:mm")
        SimpleDateFormat sdf = new SimpleDateFormat( pattern: "dd/MM/yyyy HH:mm");
        String fechaActual = sdf.format(new Date());
        nuevoHistorial.put( k: "fecha", fechaActual);
        nuevoHistorial.put( k: "usuario", email);
        // Agrega el nuevo historial con un ID único generado automáticamente
        historialRef.add(nuevoHistorial)
            .addOnSuccessListener(documentReference -> {
                // Éxito en la operación de guardado
            })
            .addOnFailureListener(e -> {
                // Error en la operación de guardado
            });
    }
}

```

guardarPlatos

Comprobamos plato a plato si existe el plato en la base de datos, de no existir se almacenará en la base de datos.

```
private void guardarPlatos(String restauranteNombre) {
    platosRef = db.collection( collectionPath: "platos");
    // Consultar los platos existentes del restaurante
    platosQuery = platosRef.whereEqualTo( field: "restaurante", restauranteNombre).whereEqualTo( field: "usuario", email);
    platosQuery.get().addOnCompleteListener(task -> {
        if (task.isSuccessful()) {
            querySnapshot = task.getResult();
            platosExistente = querySnapshot.getDocuments();
            // Guardar los nuevos platos y actualizar los existentes
            for (Plato plato : platosABd) {
                platoExistente = false;
                platoExistenteId = "";
                // Verificar si el plato ya existe en la base de datos
                for (DocumentSnapshot platoSnapshot : platosExistente) {
                    if (plato.getNombre().equals(platoSnapshot.getString( field: "nombre")))) {
                        platoExistente = true;
                        platoExistenteId = platoSnapshot.getId();
                        break;
                    }
                }
            }
        }
    })
}
```

Tras comprobar que plato existe, se actualizará en la base de datos

```
// Crear un mapa con los datos del plato
Map<String, Object> platoData = new HashMap<>();
platoData.put( k: "nombre", plato.getNombre());
platoData.put( k: "descripcion", plato.getDescripcion());
platoData.put( k: "precio", plato.getPrecio());
platoData.put( k: "imagen", plato.getUrlImage() == null ? "" : plato.getUrlImage());
platoData.put( k: "restaurante", restauranteNombre);
platoData.put( k: "usuario", currentUser.getEmail());

if (platoExistente) {
    // Actualizar el plato existente
    platosRef.document(platoExistenteId).set(platoData).addOnSuccessListener(aVoid -> {
    }).addOnFailureListener(e -> {
    });
    // Subir la imagen del plato y actualizar los datos en Firestore
    subirImagenPlato(platoExistenteId, plato.getUrlImage(), platoData);
} else {
    // Guardar el nuevo plato
    platosRef.add(platoData).addOnSuccessListener(documentReference -> {
        // Subir la imagen del plato y actualizar los datos en Firestore
        subirImagenPlato(documentReference.getId(), plato.getUrlImage(), platoData);
    }).addOnFailureListener(e -> {
    });
}
```

repartirCompartir

Cada persona posee como atributo double monedero, es el precio final que debe pagar el comensal, repartirCompartir() revisa únicamente los platos repartidos y reparte el precio entre las personas que han compartido el plato de forma equitativa

Recorrerá todas las personas, todos los platos asociados a cada persona y si es compartido (boolean esCompartido) comparará todas las personas que se encuentren dentro de la lista de getPersonasCompartir.

Para repartir el precio del plato, dividirá el precio del plato entre la lista de PersonasCompartir, que, posteriormente utilizando el método sumarMonedero() añadira su valor al monedero de cada persona.

```
//usage + Javier Calderon
public void repartirCompartido() {
    double precioPlato;
    for (int i = 0; i < comensales.size(); i++) {
        for (int j = 0; j < comensales.get(i).getPlatos().size(); j++) {
            if (comensales.get(i).getPlatos().get(j).isCompartido()) {
                comensales.get(i).getPlatos().get(j).getPersonasCompartir().add(comensales.get(i)); // ...
                precioPlato = comensales.get(i).getPlatos().get(j).getPrecio() / (comensales.get(i).getPlatos().get(j).getPersonasCompartir().size());
                for (int h = 0; h < comensales.get(i).getPlatos().get(j).getPersonasCompartir().size(); h++) {
                    for (int m = 0; m < comensales.size(); m++) {
                        if (comensales.get(m).getComensalCode() == comensales.get(i).getPlatos().get(j).getPersonasCompartir().get(h).getComensalCode()) {
                            comensales.get(m).sumarMonedero(precioPlato);
                        }
                    }
                }
            }
        }
        comensales.get(i).asignarPrecio();
    }
}
```

Finalmente, aprovechando ese mismo método, la clase Persona cuenta con un método llamado asignarPrecio que es el encargado de asignar el valor de cada plato a su persona.

```
}
```

```
//Método devuelve el total a pagar
1 usage + Javier Calderon +1
public void asignarPrecio () {
    double precioTotal=0;
    for (int i=1;i<platos.size();i++) {
        if (!platos.get(i).isCompartido()) {
            monedero+=platos.get(i).getPrecio();
        }
    }
}
```

Importar a Whatsapp

Se utiliza el método cargarHistorialWP que a parte de transformar el desglose en una cadena con saltos de línea, añade un título al mensaje Whatsapp

De no estar instalado Whatsapp, avisará con un Toast.

```
//Método que da la opción a compartir en Whatsapp
ivCompartirImagen.setOnClickListener(view -> {
    //Convierte la factura en un String
    cargarHistorialWP();

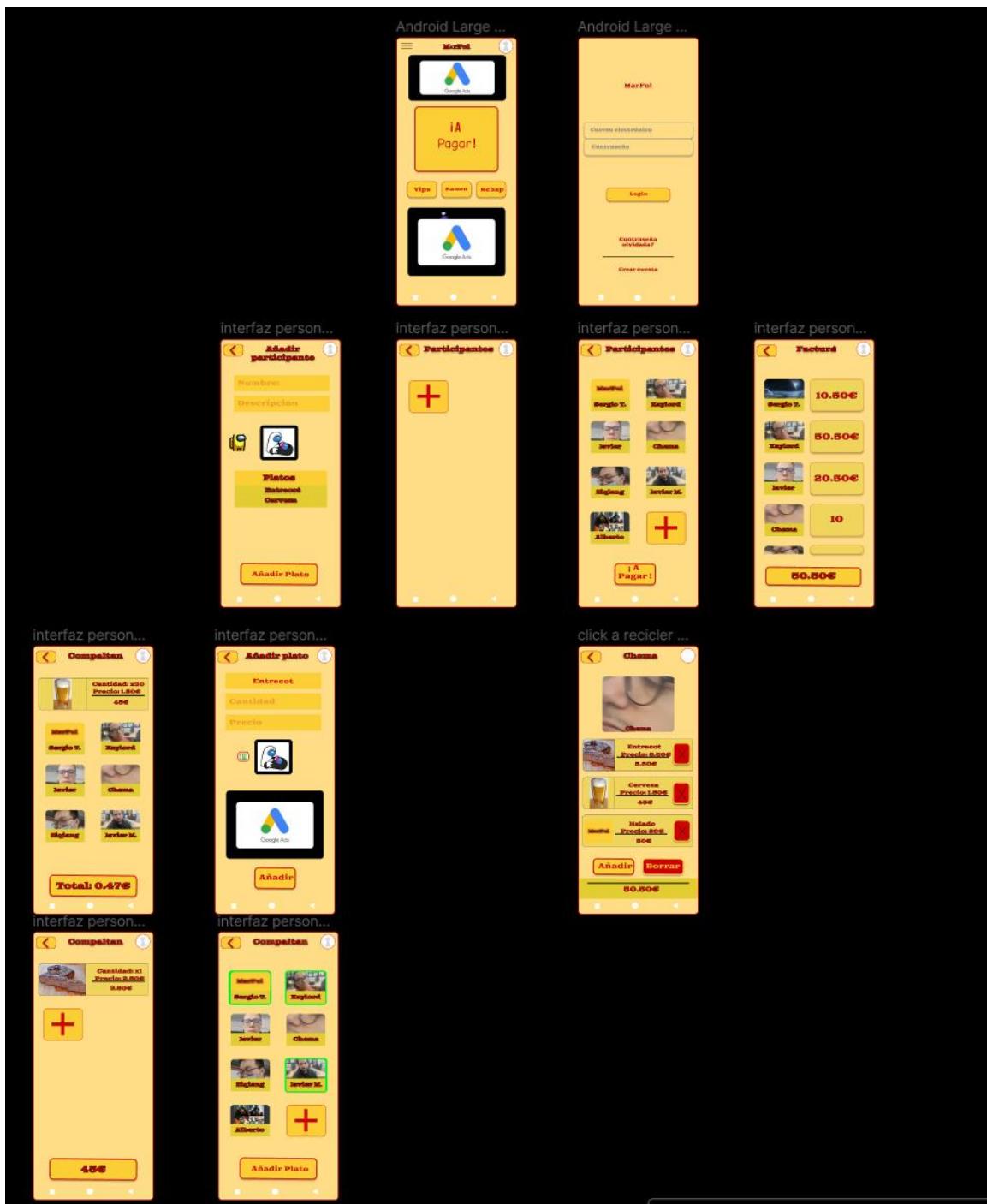
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("text/plain");
    intent.putExtra(Intent.EXTRA_TEXT, historial);
    intent.setPackage("com.whatsapp");

    try {
        startActivity(intent);
    } catch (android.content.ActivityNotFoundException ex) {
        // WhatsApp no está instalado en el dispositivo
        Toast.makeText(context, text: "WhatsApp no está instalado.", Toast.LENGTH_SHORT).show();
    }
});
```

Programación del proyecto

Definición de objetivos: El objetivo principal de Marfol es crear una aplicación móvil para simplificar y agilizar el proceso de dividir la cuenta después de una comida en un restaurante.

Planificación y diseño: Desde un principio se tenía en mente como sería Marfol, lo cual ha variado bastante desde el principio a la versión final.



Idea inicial

Selección de lenguaje y herramientas: Desde el inicio se planeó utilizar java como lenguaje principal y git como controlador de versiones.

Seguimiento:

El proyecto ha sido desarrollado por Francisco Javier López (Zharell) y Kayler Borges (Rocas), iniciado el día 24 de marzo y finalizado el día 3 de junio.

Primer y último commit

Login arreglado
 rocas • Apr 8, 2023
IndexActivity añadido, revisar recursos, utilización de drawable
 Zharell • Apr 5, 2023
Update README.ME
 Relyak • Apr 5, 2023
Create README.ME
 Relyak • Apr 5, 2023
First commit
 Zharell • Mar 24, 2023

Merge branch 'main' of https://github.com/Zharell/TFG
 Javier Calderón • 2 days ago
Añadido compartir en whatsapp
 Javier Calderón • 2 days ago

Justificación de viabilidad

- Necesidad del mercado: Existe una demanda significativa de una solución práctica y eficiente para dividir la cuenta en restaurantes. Muchas personas encuentran confusión y complicaciones al tratar de compartir los gastos de manera equitativa. Marfol aborda esta necesidad específica y proporciona una herramienta intuitiva para calcular los costos individuales de forma sencilla y justa.
- Beneficios para los usuarios: Marfol ofrece una serie de beneficios para los usuarios. Simplifica el proceso de división de la cuenta al permitir a los comensales ingresar fácilmente los platos y bebidas que han ordenado, asignando automáticamente los costos a cada individuo. Además, utiliza un algoritmo inteligente que tiene en cuenta las preferencias de los usuarios, lo que garantiza una división equitativa. También proporciona características adicionales, como la posibilidad de añadir elementos personalizados y compartir los resultados de la división de la cuenta a través de mensajes de texto. Estos beneficios hacen que Marfol sea una solución conveniente y fácil de usar.
- Potencial de mercado: El mercado objetivo de Marfol abarca a todos los usuarios que frecuentan restaurantes y desean una forma práctica y justa de dividir la cuenta. Esto incluye tanto a individuos que comen solos como a grupos de amigos, familias

y otros comensales. Dado el tamaño del mercado de restaurantes y la frecuencia con la que las personas comen fuera de casa, existe un gran potencial para captar una base de usuarios significativa.

- Tecnología y capacidad de desarrollo: Configuradores, la organización detrás de Marfol, es una empresa especializada en el diseño y desarrollo de aplicaciones móviles. Esto implica que tienen la experiencia y la capacidad técnica para desarrollar y mantener la aplicación de manera efectiva. Además, la aplicación utiliza tecnología inteligente y características adicionales que la hacen atractiva y competitiva en el mercado.

En resumen, la viabilidad de Marfol se justifica por la demanda existente en el mercado, los beneficios que ofrece a los usuarios, el potencial de mercado y la capacidad técnica de Configuradores para desarrollar y mantener la aplicación. Todos estos factores respaldan la viabilidad y el éxito del proyecto Marfol como una solución innovadora para simplificar la división de la cuenta en restaurantes.

Prevención de riesgos

- Asegurarse de tener una postura adecuada al trabajar.
- Realizar descansos regulares.
- Trabajar en un entorno bien iluminado.
- Utilizar protección auditiva al usar auriculares.
- Realizar copias de seguridad periódicas de los datos.
- Mantener el espacio de trabajo organizado.
- Gestionar el estrés de manera adecuada.
- Cuidar los ojos descansando la vista y reduciendo el brillo de la pantalla.