

DESARROLLO AVANZADO DE SOFTWARE

4º curso, grupo 46

Segundo cuatrimestre

---

## SEGUNDO PROYECTO INDIVIDUAL: BUDGET BUDDY

---

Maitane Urruela López de Echazarreta

---

Bilbao, 21 de abril de 2024

## ÍNDICE

---

# Índice

<b>1. BudgetBuddy</b>	<b>3</b>
1.1. Pantalla de Login . . . . .	3
1.2. Interfaz del usuario . . . . .	5
1.3. Pantallas . . . . .	7
1.3.1. Pantalla principal y pantallas auxiliares . . . . .	7
1.3.2. Facturas . . . . .	8
1.3.3. Dashboards . . . . .	8
1.4. Notificaciones . . . . .	8
1.5. Widget . . . . .	9
1.6. Permisos . . . . .	10
1.7. Datos . . . . .	10
<b>2. Link al repositorio</b>	<b>10</b>
<b>3. Herramientas utilizadas en el proyecto</b>	<b>11</b>
3.1. Servidor . . . . .	11
3.1.1. Google Cloud Engine . . . . .	11
3.1.2. Docker Compose . . . . .	11
3.1.3. PostgreSQL . . . . .	11
3.2. Ktor . . . . .	11
3.3. Firebase Cloud Messaging . . . . .	11
3.4. Pick Visual Media . . . . .	12
3.5. Content Providers . . . . .	13
3.6. Geolocalización . . . . .	13
3.7. Google Maps . . . . .	13
3.8. Widgets . . . . .	13
3.9. Alarm Manager . . . . .	13
<b>4. Arquitectura del proyecto y sus clases</b>	<b>15</b>
4.1. DI . . . . .	15
4.2. Utils . . . . .	15
4.3. VM . . . . .	16
4.4. Preferences . . . . .	16
4.5. Local . . . . .	17
4.5.1. Room . . . . .	17
4.5.2. DAO . . . . .	17
4.5.3. Data . . . . .	17
4.6. Repository . . . . .	17
4.7. Remote . . . . .	18
4.8. Navigation . . . . .	18

## ÍNDICE DE FIGURAS

---

4.9. Screens . . . . .	18
4.9.1. MenuScreens . . . . .	18
4.10. Shared . . . . .	18
4.11. Firebase . . . . .	19
4.12. Widgets . . . . .	19
4.13. AlarmManager . . . . .	19
<b>5. Requisitos llevados a cabo</b>	<b>20</b>
5.1. Requisitos mínimos . . . . .	20
5.1.1. Base de datos remota . . . . .	20
5.1.2. Mensajería FCM . . . . .	20
5.1.3. Captar imágenes y guardarlas en el servidor . . . . .	21
5.1.4. Uso correcto de la aplicación y pila de actividades . . . . .	22
5.2. Requisitos extra . . . . .	22
5.2.1. Content Provider . . . . .	22
5.2.2. Geolocalización . . . . .	23
5.2.3. Google Maps . . . . .	23
5.2.4. Widget . . . . .	23
5.2.5. Tarea programada mediante alarma . . . . .	23

## Índice de figuras

1. Pantallas principales de la APP. . . . .	3
2. Pantallas de inicio de sesión. . . . .	4
3. Posibles errores en el inicio de sesión y registro. . . . .	5
4. Menú lateral y sus pantallas. . . . .	6
5. Gastos desplegados y pantalla para añadir gastos. . . . .	7
6. Gastos desplegados y pantalla para añadir gastos. . . . .	9
7. Proceso de Firebase Cloud Messaging [1]. . . . .	12
8. Galería del dispositivo visualizada por PickMedia. . . . .	21
9. Eventos añadidos al calendario. . . . .	22

## 1 BUDGETBUDDY

### 1. BudgetBuddy

BudgetBuddy es una aplicación Android diseñada para facilitar el registro, visualización, descarga y compartición eficiente e intuitiva de los gastos personales. La aplicación se centra en mantener un seguimiento diario de los gastos, permitiendo a los usuarios visualizarlos según su cuantía y categoría.

Esta versión de la aplicación, se basa en la presentada en la primera entrega de la asignatura de Desarrollo Avanzado de Software, que está disponible en el siguiente link.

Respecto con la anterior entrega, la aplicación dispone de un sistema de identificación de usuarios mediante correo electrónico, que permite disponer de diferentes datos en cada usuario. Además, la personalización de la aplicación en base a idioma (inglés, castellano y euskera), tema (verde, azul y morado) y otras preferencias, también está individualizada para cada usuario (estas preferencias no se guardan de un dispositivo a otro incluso con el mismo usuario).

Al igual que en la anterior entrega, la interfaz común de la aplicación una vez iniciada la sesión se compone de un marco que abarca todas las pantallas, incluyendo las principales como “Home”, “Facturas” y “Dashboards” (Figura 1), así como varias ventanas auxiliares, que se explicarán más adelante.

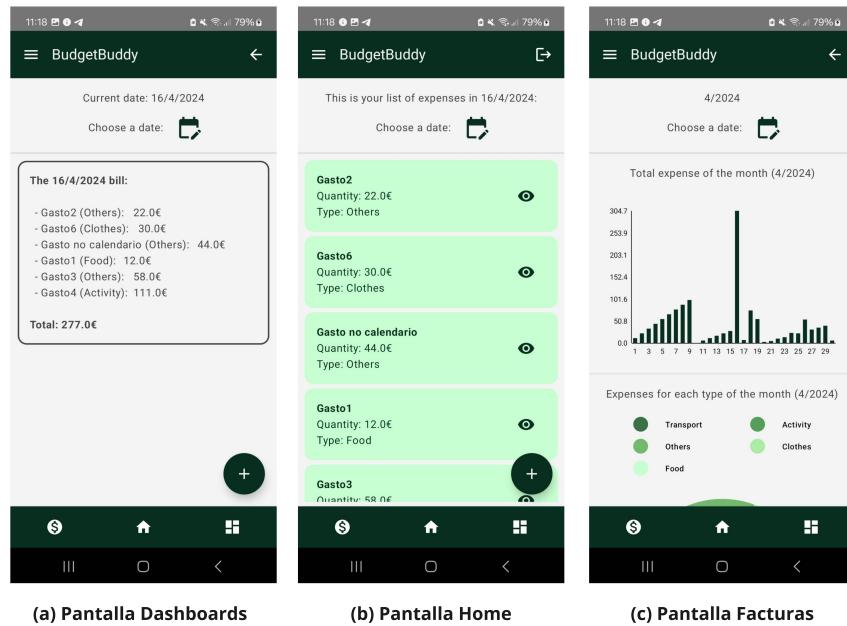


Figura 1: Pantallas principales de la APP.

#### 1.1. Pantalla de Login

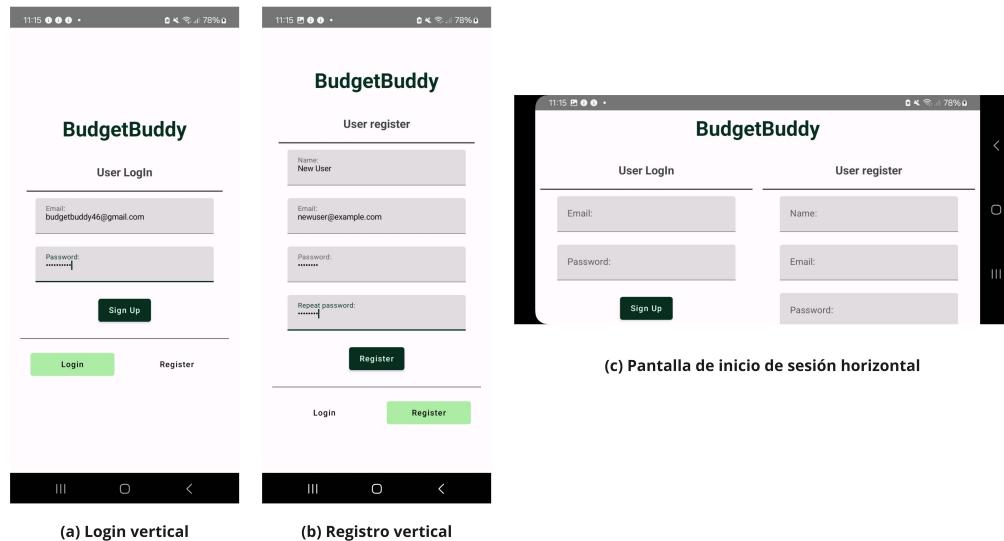
La pantalla de inicio de sesión consta de dos pantallas diferentes: la de registro (para añadir nuevos usuarios) y la de *Login* (cuando el usuario ya existía de antemano). Al estar en vertical, la

## 1 BUDGETBUDDY

navegación entre estas pantallas se hace mediante un botón inferior que da la opción de cambiar de una a otra, mientras que en formato horizontal, ambos formularios aparecen uno al lado del otro. Esta pantalla siempre aparecerá en inglés, puesto que no puede depender de las preferencias de cada usuario y es el idioma estándar de la APP.

Los requisitos para registrarse en la aplicación son los siguientes:

- **Nombre de usuario:** El nombre no puede estar vacío. Además de eso no se requiere de nada más, puesto que no se usa para identificar al usuario de ninguna forma. Es decir, dos usuarios pueden tener el mismo nombre sin problema.
- **Correo electrónico:** El correo electrónico tiene que tener un formato válido, es decir: <usuario>@<dominio>. <extensión>. No es necesario que esté registrado en Google ni nada por el estilo, pero sí conviene que sea un correo válido y a poder ser que esté sincronizado con el teléfono de cara al uso de la APP (esto queda a elección del usuario). A diferencia del anterior campo, este se usará para identificar al usuario, por lo que no puede estar repetido.
- **Contraseña:** Al igual que el nombre de usuario, no tiene restricciones salvo la de estar vacía. A la hora de registrarse, los dos campos de la contraseña deben coincidir. Todo lo demás (caracteres usados y la seguridad general que ofrezca la contraseña) queda a elección del usuario. Esta contraseña será convertida en *hash* antes de ser guardada en la base de datos, por lo que la aplicación le dará seguridad extra por defecto.



**Figura 2:** Pantallas de inicio de sesión.

En caso de no haber añadido uno de los campos correctamente aparecerá un mensaje de error indicándolo. Además, si no hay conexión a Internet o hay problemas para conectar con el servidor, también aparecerá un mensaje informando de ello. Por último, **un usuario no puede estar conectado en dos dispositivos a la vez** por lo que, si el usuario ya está conectado en otro

## 1 BUDGETBUDDY

dispositivo, habrá que cerrar sesión mediante el botón de la pantalla principal de este antes de iniciarla en otro sitio.

**Nota:** La única manera de hacer *logout* es pulsando el botón de salida de la pantalla "Home". Al cerrar la aplicación se vuelve a pedir el nombre de usuario y la contraseña por seguridad, pero la sesión seguirá iniciada hasta que se pulse el botón.

Esto es importante a tener en cuenta antes de desinstalar la aplicación de un dispositivo, ya que los datos de la APP se borrarán del teléfono y es probable que no se pueda recuperar la cuenta (en cuyo caso habría que pedir el desbloqueo de la cuenta al servicio técnico de BudgetBuddy: budgetbuddy46@gmail.com).

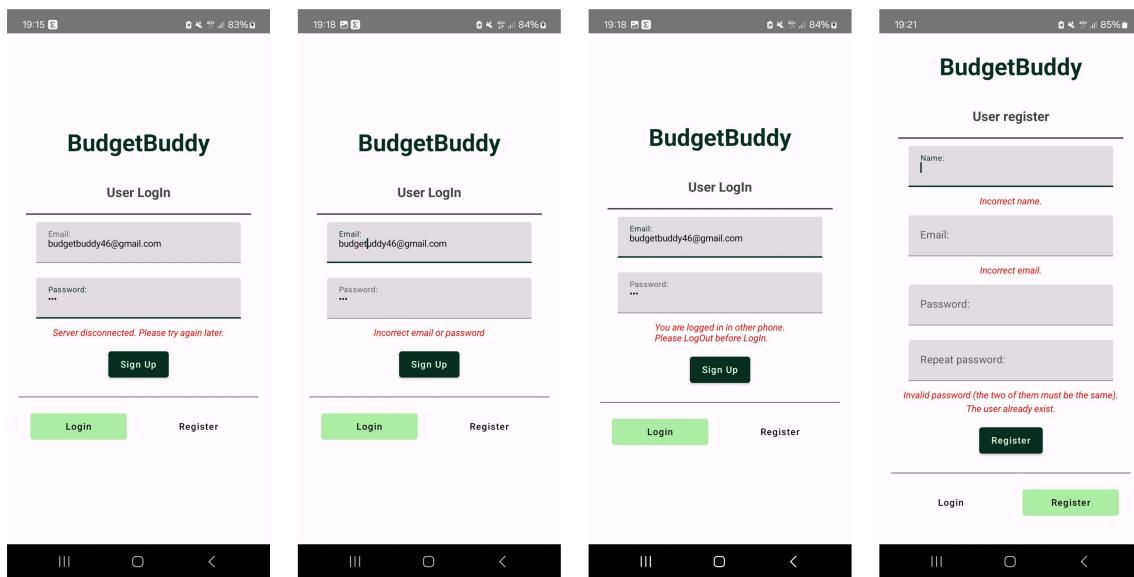


Figura 3: Posibles errores en el inicio de sesión y registro.

### 1.2. Interfaz del usuario

Una vez realizado el inicio de sesión, se abre el marco común de todas las pantallas: un *Scaffold* que dispone de una barra superior y otra inferior, que en caso de estar en horizontal la pantalla, estas se convierten en una barra superior y una lateral izquierda, respectivamente.

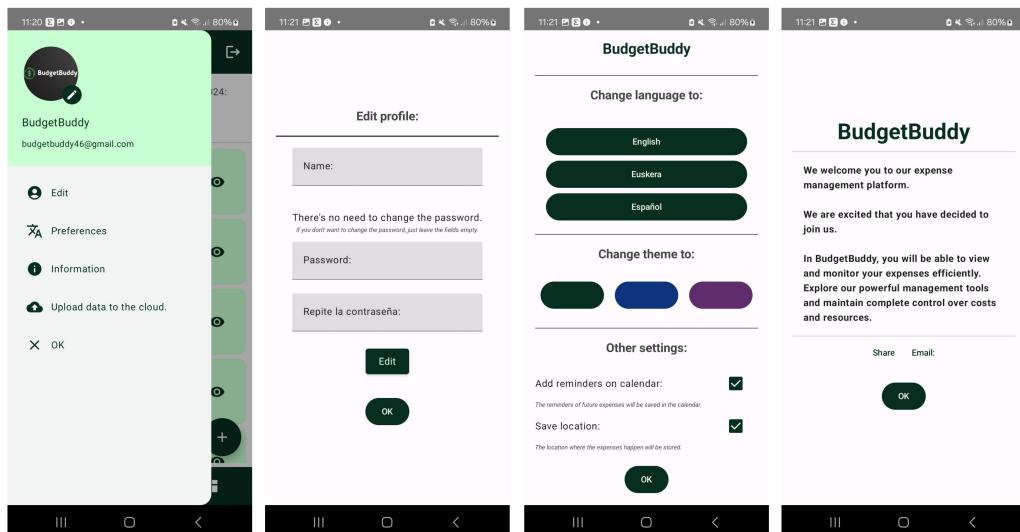
La barra superior cuenta con el nombre de la aplicación y una flecha para regresar a la pantalla principal (a menos que el usuario ya esté en ella, en cuyo caso será el botón de *Logout*). Además, cuenta con un botón para abrir el menú lateral izquierdo que contiene:

- **Perfil del usuario:** En este aparece la foto de perfil del usuario con un botón para editarla (por defecto aparecerá un ícono estándar), el nombre y el correo electrónico del usuario.
- **Panel de navegación,** con las opciones para navegar a las diferentes pantallas:

## 1 BUDGETBUDDY

---

- **Edición de usuario:** En esta pantalla se pueden editar tanto el nombre como la contraseña del usuario (en caso de solo querer cambiar el nombre, no es necesario poner la contraseña).
- **Preferencias:** En esta pantalla cada usuario elige el idioma y tema de colores en el que desea visualizar la aplicación, además de seleccionar si quiere guardar los eventos futuros (gastos futuros) en el calendario y guardar la localización de cada uno de estos. Todas estas preferencias se almacenan en el *DataStore* del dispositivo, por lo que al cambiar de teléfono, habrá que volver a especificar las preferencias. Por defecto la aplicación se visualiza en inglés con el tema verde y se guarda toda la información.
- **Información:** En esta pantalla se da información general de la APP, con la opción de realizar dos acciones: “Compartir” y “Email”. La primera opción permite enviar una invitación a otro usuario a través del canal de chat de elección del usuario, mientras que la segunda abre el correo electrónico predeterminado del usuario con una plantilla dirigida al correo de BudgetBuddy (budgetbuddy46@gmail.com).
- **Subir datos a la nube:** Por defecto, al cerrar sesión se almacenan todos los datos en la nube, pero mediante este botón se da la opción de hacer otra copia de seguridad previa.
- **Cerrar:** Pulsando este botón se cierra el menú lateral (también es posible abrirlo y cerrarlo desplazando la pantalla hacia la derecha e izquierda).



**Figura 4:** Menú lateral y sus pantallas.

En la barra inferior (o lateral en su defecto), aparecerán los iconos de las tres pantallas principales de “Home”, “Facturas” y “Dashboards” (Figura 1), de forma que al seleccionar uno de ellos, se muestra la pantalla correspondiente al usuario. En el modo horizontal, se agrega un cuarto ícono de “Agregar” a la barra de navegación, haciendo desaparecer el botón flotante con la misma función de la pantalla inicial.

Además de estas secciones, estas tres pantallas cuentan con un encabezado que permite al usuario cambiar la fecha en la que se encuentra. Este abre el calendario con la fecha por defecto del día actual (al no guardarla en las preferencias, al iniciar la aplicación siempre sale el día de la fecha actual). Más tarde el usuario podrá cambiarla para realizar gestiones de otro día, o visualizar los datos referentes a este.

### 1.3. Pantallas

#### 1.3.1. Pantalla principal y pantallas auxiliares

La primera pantalla principal, conocida como “Home” (Figura 1), muestra todos los gastos realizados en un día a elección del usuario. Cada gasto se muestra en un recuadro aparte, visualizando su nombre, la cantidad asociada a este y el tipo de gasto al que pertenece. Además, cada uno de ellos cuenta con un botón que permite visualizar todos los datos del gasto y las opciones que ofrece (borrado y edición).

Al abrir el gasto, además de la información mencionada, se mostrará la localización donde se hizo en caso de que hubiese (recordemos que se puede deshabilitar esta función), en caso contrario se mostrará un ícono de un mapa y un recordatorio de que se debe habilitar esa función para más información.

Mediante los botones de la esquina superior derecha, se podrá borrar el gasto o editarlo. En caso de realizar esta última acción, se dará la opción a cambiar cualquier campo del gasto, incluida la fecha en la que se produjo. Además se actualizará la ubicación del gasto a aquella en la que se encuentre el usuario en ese momento. Tanto al borrar como al editar un elemento con éxito, aparecerá un mensaje en la parte inferior.

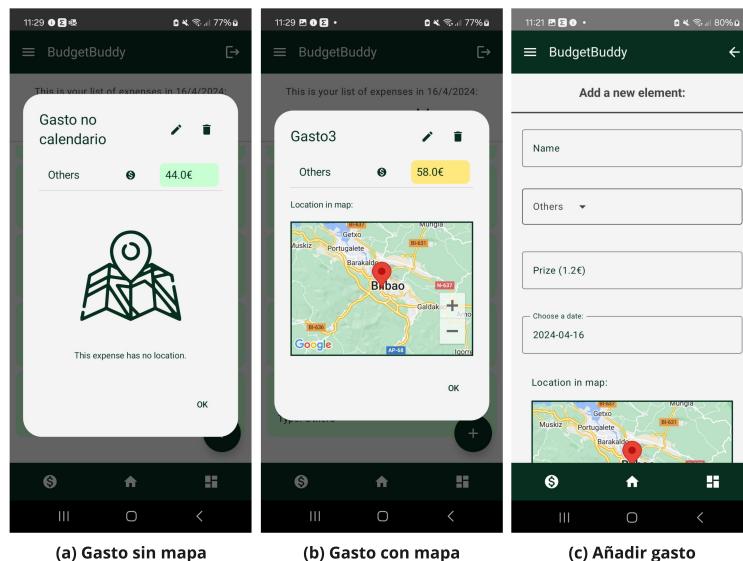


Figura 5: Gastos desplegados y pantalla para añadir gastos.

Como característica especial, esta pantalla cuenta con un botón flotante con el símbolo de “Añadir”, que al ser pulsado lleva al usuario a un formulario donde meter los datos de un nuevo gasto:

- **Nombre:** El nombre del gasto o una explicación de este en su defecto.
- **Tipo:** El tipo de gasto que es (por defecto a “Otros”).
- **Cantidad:** La cantidad en euros que ha supuesto ese gasto (en caso de querer añadir decimales se separa por puntos).
- **Fecha:** Da la opción de escoger la fecha del gasto mediante un calendario desplegable (por defecto con la fecha que venía seleccionada en la pantalla principal).
- **Ubicación:** Aparecerá un mapa con la geolocalización actual, en este caso no es algo editable.

En caso de no introducir algo en todos los campos, aparecerá una alerta, del mismo modo que al introducir el campo de la cantidad con una coma (u otro signo de puntuación) en vez de un punto. Esta pantalla sigue la misma estructura que la de edición (Figura 5).

Si el gasto es de una fecha futura y todos los permisos están activos, se añadirá un evento en el calendario con los datos del gasto, además de programar una alarma para el día en el que se añade como recordatorio de la existencia de ese gasto.

### 1.3.2. Facturas

En esta pantalla (Figura 1), aparece la factura asociada al día seleccionado. La factura cuenta con un encabezado con la fecha, un listado de todos los gastos, y la suma de todos estos al final. Como punto destacable de la pantalla y al igual que la anterior, cuenta con un botón flotante, en este caso desplegable, dando lugar a dos acciones posibles: “Descargar” y “Compartir”.

Al descargar el fichero, se generará una notificación en el dispositivo, además de un fichero TXT en el directorio de descargas del teléfono. En cambio, al compartir, se enviará el contenido de la factura por el canal de mensajería de elección del usuario, al contacto seleccionado.

### 1.3.3. Dashboards

En esta última pantalla (Figura 1), al igual que en las dos anteriores, se permite al usuario seleccionar una fecha. Sin embargo, en este caso se muestra información relativa a todo el mes seleccionado. Un gráfico de barras ilustra el gasto diario del mes, acompañado por una gráfica de donut que representa el porcentaje asociado a cada tipo de gasto en el mes completo.

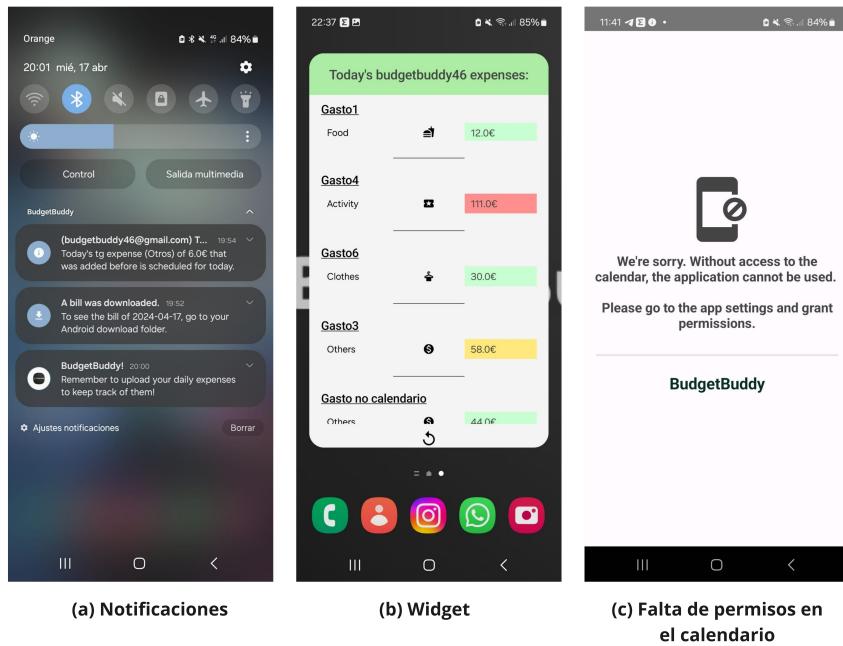
## 1.4. Notificaciones

Hay varios tipos de notificación en la aplicación. Por un lado están las notificaciones creadas al momento como la que se crea al descargar el fichero de la factura en la pantalla “Facturas”. Esta te avisa de que se ha descargado correctamente el fichero.

## 1 BUDGETBUDDY

Por otro lado están las通知aciones programadas (Figura 6). Estas se crean al añadir un gasto futuro en la aplicación y se programan para el día establecido mediante el *AlarmManager* (a las 11:00am).

Por último están los mensajes de difusión mediante Firebase que llegan a todos los usuarios. Concretamente hay uno programado para las 11:00am y 8:00pm de cada día que recibirán todos los usuarios con la aplicación instalada, a modo de recordatorio de hacer uso de la APP. En caso de estar utilizando la aplicación en el momento no aparecerá, ya que la aplicación tiene que estar cerrada para que aparezca (Figura 6).



**Figura 6:** Gastos desplegados y pantalla para añadir gastos.

### 1.5. Widget

La aplicación dispone de un *widget* que muestra una lista con los gastos del día de hoy del usuario que tiene la sesión iniciada en la APP (de no tener, la lista aparecerá vacía). En caso de que haya más de una sesión iniciada, se mostrarán los del último usuario registrado y en caso de que no haya ninguno registrado, el *widget* mismo pedirá el inicio de sesión.

Por cada gasto se mostrará su nombre y debajo de este el tipo de gasto al que pertenece, un ícono que represente este tipo y la cantidad que ha supuesto el gasto. Además, si el gasto es de entre 0€ y 50€ aparecerá en verde; de entre 50€ y 100€ aparecerá en amarillo y en caso de superar los 100€ estará subrayado en rojo (Figura 6).

## 2 LINK AL REPOSITORIO

---

### 1.6. Permisos

Para el correcto funcionamiento de esta aplicación son necesarios varios permisos. En primer lugar es estrictamente necesario que la aplicación tenga permisos para acceder al calendario, ya que en caso de no tener este permiso, se procederá a impedir el uso de la aplicación (Figura 6).

Todos los demás permisos no son estrictamente necesarios para que la APP funcione, pero sí que limitan sus funcionalidades. Es decir, en caso de no haber permisos de notificaciones, estas no llegarán, o al no haber permisos de localización, los mapas quedarán ocultos. Sin embargo, la falta de estos permisos no supone ningún problema para el uso general de la aplicación. Lo mismo pasa con los permisos de almacenamiento: no se descargarán ficheros, pero no dejará de funcionar la aplicación.

### 1.7. Datos

Hay un único usuario de prueba en la aplicación con todos los datos añadidos. A este usuario se accede con los siguientes datos:

- **Usuario:** budgetbuddy46@gmail.com
- **Contraseña:** 123

## 2. Link al repositorio

Esta versión de la aplicación de BudgetBuddy se encuentra en el siguiente *link*:

[https://github.com/Maits27/DAS\\_BudgetBuddy\\_2](https://github.com/Maits27/DAS_BudgetBuddy_2)

La API correspondiente en el siguiente *link*:

<https://github.com/Maits27/BudgetBuddyAPI>

Y al servidor con la base de datos remota se accede mediante el siguiente comando:

ssh -i /path/a/clave/privada <usuario>@34.135.202.124

### 3 HERRAMIENTAS UTILIZADAS EN EL PROYECTO

---

## 3. Herramientas utilizadas en el proyecto

### 3.1. Servidor

#### 3.1.1. Google Cloud Engine

Google Cloud Engine ofrece la posibilidad de crear y ejecutar máquinas virtuales en línea de forma sencilla, proporcionando una infraestructura en la nube confiable y de alto rendimiento [2]. En este proyecto se ha utilizado un servidor en Google Cloud para almacenar el contenedor Docker con la base de datos remota que guarda la información de todos los usuarios registrados en la aplicación y sus respectivos gastos.

#### 3.1.2. Docker Compose

Docker es un sistema operativo para contenedores que virtualiza el sistema operativo de un servidor, similar a cómo una máquina virtual virtualiza el hardware del servidor. Mediante Docker Compose, se han relacionado dos contenedores de Docker, uno para la base de datos, utilizando la imagen base de PostgreSQL y otra para la API de FastAPI que conecta con la base de datos.

FastAPI es un *framework* para crear APIs utilizando Python. Este *framework* funciona con cualquier base de datos y con cualquier librería que sirva para conectarse a estas. En este caso se ha empleado SQLAlchemy como una librería ORM (Mapeo Objeto-Relacional), evitando la necesidad de escribir consultas SQL manualmente, similar a como se hizo con ROOM en la primera entrega.

Esto se logra creando clases que representan las tablas de la base de datos y utilizando estas clases para interactuar con las tablas reales. Además de obtener información de las tablas sin necesidad de escribir consultas, SQLAlchemy también ofrece herramientas para establecer relaciones entre las diferentes tablas [8].

#### 3.1.3. PostgreSQL

El sistema de gestión de bases de datos utilizado en esta aplicación ha sido PostgreSQL, ya que es uno de los sistemas compatibles con SQLAlchemy.

### 3.2. Ktor

Ktor es un *framework* ligero basado en Kotlin utilizado para crear aplicaciones y servicios web en el lado del servidor. Proporciona una API simple y flexible para construir aplicaciones asíncronas, basadas en eventos y no bloqueantes [10]. Este *framework* se ha utilizado para realizar las peticiones necesarias a la API del servidor de Google Cloud que conecta con la base de datos remota.

### 3.3. Firebase Cloud Messaging

Firebase es una plataforma de desarrollo de aplicaciones móviles y web, así como una plataforma de *backend* como servicio (BaaS) desarrollada por Google. En el proyecto, se ha utilizado Firebase

### 3 HERRAMIENTAS UTILIZADAS EN EL PROYECTO

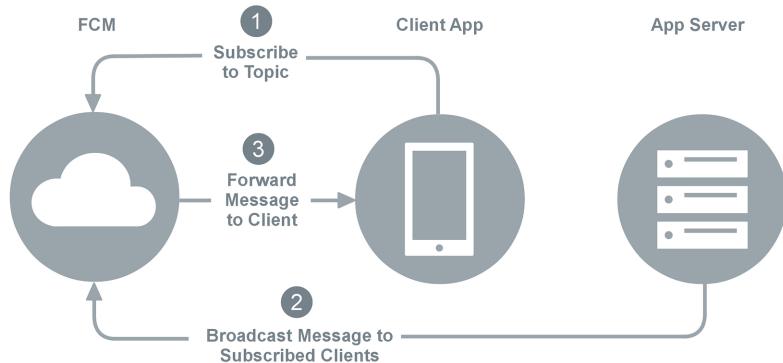
---

Cloud Messaging, que ofrece dos maneras de enviar un mismo mensaje a varios dispositivos a la vez [9]:

- **Mensajes por tema:** Permiten enviar un mensaje a múltiples dispositivos que hayan aceptado un tema específico.
- **Mensajes por grupos de dispositivos:** Permiten enviar un mensaje a los dispositivos de un grupo definido por el desarrollador.

En este caso, se ha utilizado la segunda forma, puesto que al registrarse en la aplicación, se añade automáticamente el dispositivo a la lista de difusión de Firebase de la aplicación de BudgetBuddy denominada “all”.

El token de cada dispositivo registrado se almacena en el mismo servidor que la base de datos, haciendo uso de la librería *firebase\_admin*. Al agregar el documento de credenciales proporcionado por Firebase, esta librería gestiona los tokens de los dispositivos al registrarlos y más adelante al recibir la solicitud (a través de FastAPI en este caso) para enviar la notificación a un tema concreto (Figura 7).



**Figura 7:** Proceso de Firebase Cloud Messaging [1].

#### 3.4. Pick Visual Media

Para seleccionar fotos de la galería se ha utilizado el *framework* del selector de fotos que hereda de *ActivityResultContract* y utiliza el *MediaStore.ACTION\_PICK\_IMAGES* para recoger una o múltiples imágenes. Es decir, utiliza un *ContentProvider* de manera indirecta.

El selector de fotos ofrece una interfaz de búsqueda explorable que muestra al usuario su galería de fotos ordenada por fecha (de lo más reciente a lo más antiguo). En caso de que el usuario tenga *ContentProviders* multimedia en la nube aptos en su dispositivo también puede seleccionar las fotos y vídeos almacenados de forma remota [7].

### 3 HERRAMIENTAS UTILIZADAS EN EL PROYECTO

---

#### 3.5. Content Providers

Los Content Provider son interfaces estándares que almacenan datos tanto del sistema operativo como de otras aplicaciones y permiten que las aplicaciones accedan a ellos. Los proveedores que ofrece la plataforma de Android exponen estos datos como un conjunto de tablas basadas en un modelo de base de datos relacional, en el que cada fila representa un elemento y las columnas sus características. En este caso se ha utilizado el Calendar Provider siendo este un repositorio de los eventos de calendario de un usuario [6].

Habitualmente, se utilizan *intents* implícitos para dirigir al usuario al calendario con los datos relevantes, permitiéndole agregar eventos manualmente. Sin embargo, también es posible interactuar directamente con el Calendar Provider mediante consultas y URIs específicas, como es el caso.

#### 3.6. Geolocalización

Para saber la ubicación actual del usuario se ha hecho uso del servicio de Google Play de Fused-LocationProviderClient. Este permite recoger la última localización conocida del usuario mediante el método *lastLocation* que en este caso hace uso de los permisos de ACCESS\_FINE\_LOCATION [5]. Esta localización actualiza un *Flow* de tipo *Location* dentro del *AppViewModel* al entrar en las pantallas de añadir y editar gasto.

Una vez recogida la ubicación actual, esta puede ser guardada en la base de datos (tanto local como remota) en formato de latitud y longitud por separado, ya que ninguna de las bases de datos aceptan datos de tipo *Location*, pero sí de tipo *Float* o *Double*.

#### 3.7. Google Maps

Tanto al recoger la ubicación actual, como al recoger la ubicación de un gasto en la base de datos, esta se visualiza mediante el mapa de Google Maps que proporciona Google Maps Platform, permitiendo mostrar la localización que se le pase [11].

#### 3.8. Widgets

Los Widget son complementos de la aplicación que se pueden mostrar tanto en el escritorio como en la pantalla de bloqueo, pudiendo ser informativos, de colección o de control (o una mezcla de estos). En Jetpack Compose, los Widget heredan de GlanceAppWidget, disponiendo de pocas funcionalidades, ya que la librería continúa en la primera versión y sigue en desarrollo [4].

#### 3.9. Alarm Manager

Esta clase proporciona acceso a los servicios de alarma del sistema, que permiten programar una ejecución a elección del desarrollador en algún momento en el futuro. Al sonar una alarma, el *intent* registrado para ella es transmitido por el sistema, iniciando automáticamente la aplicación objetivo si aún no está en ejecución. Las alarmas registradas se mantienen mientras el dispositivo

### 3 HERRAMIENTAS UTILIZADAS EN EL PROYECTO

---

está dormido (y pueden despertar opcionalmente al dispositivo si suenan durante ese tiempo), pero se borrarán si se apaga, reinicia o desinstala la aplicación.

El Alarm Manager está destinado a casos en los que el código de la aplicación deba ejecutarse en un momento específico, incluso si esta no está en ejecución en ese momento. Para operaciones de temporización normales (*ticks*, tiempos de espera, etc.), es más eficiente el uso de Handler [3].

## 4. Arquitectura del proyecto y sus clases

En el directorio raíz, solo se encuentran tres archivos que son *BudgetBuddyApp.kt*, *MainActivity.kt* y *MyApp*.

El primero de esos ficheros, contiene la clase **BudgetBuddyApp**, que hereda de **Application**. A simple vista, el fichero es irrelevante para el funcionamiento de la aplicación, debido a que se encuentra prácticamente vacío. Sin embargo, es totalmente necesario para el correcto funcionamiento de la librería Hilt.

Por otro lado, en la clase **MainActivity** del segundo fichero (que hereda de **AppCompatActivity**), se define la actividad principal que carga la interfaz. Aquí se crean los *ViewModel* de usuario, aplicación y preferencias, se solicitan los permisos de notificación y calendario (en caso de no estar ya dados) y se crea el canal de notificaciones correspondiente.

También se crean, por un lado el cliente de servicios de localización, que se usará en toda la APP para geolocalizar el dispositivo y el **PickVisualMedia** que servirá para seleccionar la foto de perfil. Además, se ha añadido la función para descargar ficheros, que también requiere de solicitar permisos de descarga y almacenamiento.

Por último, en **MyApp** se define el *NavHost* principal de la aplicación, que navega entre la pantalla de inicio de sesión y el contenido de la aplicación.

Los demás ficheros se han almacenado en diferentes carpetas.

### 4.1. DI

Esta carpeta de *Dependency Injection*, contiene el único fichero de  *AppModule.kt* con el objeto (*singleton*) de  **AppModule**. Al igual que la clase de tipo  **Application**, este objeto lo utiliza Hilt para definir los objetos (interfaces, DAOs, Widget...) que se deben inyectar en otras clases y cómo se deben inyectar.

Todos estos irán etiquetados como *@Singleton*, ya que solo debe existir una instancia de cada uno. Al estar a nivel de aplicación, no se destruirán hasta que la aplicación se cierre por completo.

### 4.2. Utils

Esta carpeta contiene: *TypeChange.kt*, *LanguageConfigUtils.kt*, *Permissions*, *Userverification* y *ContentProvider*.

En el primero se ha implementado la conversión de *LocalDate* a *Long* y viceversa (no confundir con los *TypeConverter* que son utilizados por Room Database). Además se implementan métodos para *Hashear Strings* y comparar *Hashes* entre ellos. También se hace la conversión entre los tipos *LatLng* y *Location* y la conversión entre los tipos de datos que se insertan en la base de datos local y la del servidor.

En el segundo fichero, se ha creado la clase *singleton* de **LanguageManager** que se encarga de cambiar el idioma de la aplicación Android.

En el fichero *Permissions* se han implementado todos los composable necesarios para la petición de permisos (calendario, localización, notificaciones, almacenamiento...).

En **UserVerification** se hacen las comprobaciones necesarias para dar el visto bueno a los datos añadidos por el usuario a la hora de realizar el registro de una nueva cuenta.

Por último, en **ContentProvider** se implementa el uso de el proveedor de calendario, necesario a la hora de agregar eventos a estos. Cuenta con métodos para coger los ID de los calendarios válidos (los disponibles localmente) y para filtrarlos en base a su validez a la hora de añadir los gastos (por ejemplo, si pertenecen a un correo diferente al de la cuenta del usuario, no se añadirá el gasto en estos).

#### 4.3. VM

Esta carpeta contiene los *ViewModel* tanto de los datos de la aplicación como de los usuarios y las preferencias, mediante las clases **AppViewModel** **UserViewModel** y **PreferencesViewModel**, todas heredando de **ViewModel** y siendo etiquetadas como *@HiltViewModel*, para que el *framework* sea capaz de injectar las interfaces de los repositorios que conectan estas clases con Room y Datastore.

El *ViewModel* de la APP, contiene todos los *Flows* de los datos de cada usuario (sus gastos), necesarios para el correcto funcionamiento de la APP.

Por otro lado, el **UserViewModel** se encarga de las mismas actualizaciones que el anterior, solo que con los datos de los usuarios. Sin embargo, en este caso y al ser todas las comprobaciones y cambios directamente con la base de datos en remoto, no utiliza flujos sino valores estáticos.

De la misma manera que el **AppViewModel**, el **PreferencesViewModel** recoge los flujos de los valores del idioma (tipo *AppLanguage*), el tema (tipo *Int*) y las diferentes preferencias de guardado (tipo *Boolean*) y a la hora de utilizarlos, recoge el último valor registrado en el flujo. Todos estos valores dependen de el último usuario registrado, que también queda guardado en este *ViewModel*. En caso de suceder alguna alteración, el cambio sería inmediato.

#### 4.4. Preferences

Esta carpeta guarda los ficheros de *IGeneralPreferences.kt*, que es la interfaz del *Repository* utilizado para la conexión con el Datastore y *PreferencesRepository.kt*, el fichero que recoge la implementación del repositorio.

## 4.5. Local

En esta carpeta se guardan por subcarpetas diferentes tipos de fichero, todos con el objetivo del correcto funcionamiento de Room y la implementación de los diferentes tipos de datos utilizados en toda la APP (de tipo *enumeration* o *data class*).

### 4.5.1. Room

Entre los ficheros de Room se encuentran, la clase de la base de datos que hereda de **RoomDatabase (Database)** y recoge como entidades las clases **Gasto** y **User**. Estas entidades se encuentran en el fichero de *DataTables.kt* marcadas con `@Entity`.

Dentro de **Gasto**, se utiliza un tipo de dato *LocalDate*, que la base de datos no reconoce. Es por esto que en el fichero *Converter.kt* (fuera de esta carpeta, en la carpeta *Local*) se implementa la clase **Converters** con dos funciones etiquetadas con `@TypeConverter` para pasar este tipo de dato a *Long* y viceversa de forma automática.

### 4.5.2. DAO

Con la entidad **Gasto** interactúa la interfaz del DAO (**GastoDao**) que se recoge en el fichero *GastoDao.kt* de esta carpeta. En esta interfaz se definen todas las consultas que se pueden realizar a la base de datos (no se podrá realizar ningún tipo de consulta que no aparezca aquí recogida). Finalmente, este DAO se llama desde la clase singleton **GastoRepository** (sección 4.6). Se sigue la misma estructura para el *UserDao.kt*.

### 4.5.3. Data

En el fichero *DataClasses* de esta carpeta recogen los *data class* de **GastoDia**, **GastoTipo**, **AlarmItem** y **Diseño**, que se usan en los flujos de los composable de *MainView* y *Dashboards* (sección 4.9). También se implementan los *data class* de tipo `@Serializable` de **AuthUser**, **PostGasto** y **CompactGasto** que se utilizan para pasarles estos datos tanto a la base de datos remota como al *Widget*.

Además se recogen tres ficheros *AppLanguage* y *TipoGasto.kt* que implementan cada uno la clase **Enumeration** con el nombre del fichero y un método que devuelve el nombre del objeto en el idioma de preferencia.

## 4.6. Repository

En esta carpeta aparecen, por un lado, la clase singleton **GastoRepository** del fichero *GastoRepository.kt* (donde se inyecta mediante Hilt la interfaz correspondiente al repositorio, que ha sido definida en el mismo fichero).

Lo mismo ocurre con *UserRepository.kt*, que además implementa la interfaz que se encuentra en la misma carpeta: **ILoginSettings**. Esta última se encarga de la gestión del último usuario registrado en la aplicación, y por lo tanto de conectar los repositorios de preferencias y usuario.

## 4.7. Remote

En esta carpeta se guarda el cliente HTTP que implementa los métodos de la librería Ktor para comunicarse con la API del servidor y por ende, con la base de datos en remoto.

## 4.8. Navigation

Aquí se encuentra un único fichero llamado *AppScreens* que contiene una ***sealed class*** con el mismo nombre. Dentro se recogen todas las pantallas a las que se puede acceder desde el *NavHost* y sus nombres (sección 4.9).

## 4.9. Screens

En esta carpeta se guardan todos los ficheros con las pantallas a las que se puede acceder desde el *NavHost* (“Login”, “App”, “Mainview”, “Home”, “Dashboards”, “Facturas”, “Add” y “Edit”).

El *NavHost* principal cambia entre las pantallas de **Login** y **App** dependiendo del inicio de sesión. Esta última contiene el menú de navegación lateral (sección 4.9.1) y la visualización del **MainView**.

El **MainView** es el composable principal de la APP, que recoge el *Scaffold* que define la estructura principal de la APP. En la *trailing lambda*, del *Scaffold* (su contenido) se ha añadido directamente otro *NavHost* de forma que, dependiendo de la ruta que se le proporcione, se visualice el composable correspondiente a esta (cambia entre las tres pantallas principales).

Los demás ficheros de la carpeta tienen los composable relativos a la pantalla que representan. En caso de tener alguno de estos elementos en común entre las pantallas, este se encontraría en la carpeta “Shared”.

### 4.9.1. MenuScreens

En esta carpeta se han agregado las pantallas que se usan únicamente en el *NavHost* del menú desplegable lateral. Estas pantallas están compuestas por los composable de **Infor**, **Preferences** y **UserEdit**.

## 4.10. Shared

En esta carpeta se guardan todos los composable y alertas comunes entre las pantallas, además de las notificaciones e *intents* implícitos.

Por un lado, en el fichero *Comunes.kt* se implementan las funciones como **NoData** (pantalla que aparece ante la falta de datos), **Calendario**, **MapScreen** o **Header**, entre otras, que aparecen en varias de las pantallas principales.

Los **AlertDialog** como las alertas de errores, se definen en *Alerts.kt*. También guarda el **Toast** que se imprime en caso de edición o borrado de un elemento. Estos composable se han separado de los anteriores por su naturaleza de aviso, pero son usados de la misma manera.

En *Style.kt* se guardan los composable que tienen más que ver con el estilo que se le da a ciertos elementos de la APP, como el texto en formato de: título, subtítulo o descripción.

Por último, en *Notifications.kt* se guarda la implementación de la notificación en caso de descarga. Además de una función que sirve para compartir contenido mediante *intents* implícitos.

#### 4.11. Firebase

Esta carpeta contiene la clase **MyFirebaseMessagingService** que hereda de *FirebaseMessagingService*. En esta clase se define el proceso a realizar al recibir una notificación del servicio Firebase.

#### 4.12. Widgets

En esta carpeta se definen las clases **WidgetReceiver** y **Widget**. El primero se encarga de relacionarse con los datos de la aplicación mediante *Hilt* de forma que reciba los datos necesarios y pueda actualizar el *Widget* cuando sea necesario, mientras que el segundo implementa el contenido del elemento y visualiza esos datos.

#### 4.13. AlarmManager

Por último, en esta carpeta se recogen tres elementos. **AlarmScheduler**, una interfaz que recoge los métodos de crear y cancelar una alarma. **AndroidAlarmScheduler**, que implementa la interfaz especificando que hacer al registrar y cancelar una alarma y por último **AlarmReceiver**, un *BroadcastReceiver* que al recibir la alarma programada ejecuta el código necesario (en este caso crea una notificación).

## 5. Requisitos llevados a cabo

### 5.1. Requisitos mínimos

#### 5.1.1. Base de datos remota

Como se explica sobre el servidor la base de datos se ha creado haciendo uso de Docker Compose para conectar los contenedores de PostgreSQL (base de datos) y FastAPI (API). Esta configuración se puede encontrar en el siguiente GitHub: <https://github.com/Maitis27/BudgetBuddyAPI.git>. Además, será posible conectarse al servidor haciendo uso de claves SSH utilizando el siguiente comando: ssh -i /path/a/clave/privada <usuario>@34.135.202.124. Los ficheros que aparecen en el GitHub se encuentran en el directorio **/home/maitane/BudgetBuddyAPI** del servidor.

Como se ha explicado antes, dentro de la aplicación, todos los métodos que conectan directamente con el servidor se encuentran en la carpeta “Remote” en el fichero *HTTP.kt*. Los métodos que conectan con los de este fichero se encuentran repartidos entre los diferentes repositorios.

En la base de datos se guarda toda la información de la aplicación. Más concretamente, los cambios en los usuarios son inmediatos, puesto que rara vez ocurren y son rápidos de gestionar al ser pocos datos, mientras que los cambios en los gastos se realizan de golpe al pulsar el botón de copia de seguridad o al hacer *Logout* con el usuario. Cada vez que un usuario haga *Login* después de haber cerrado la sesión, también se descargarán los datos de este, tardando unos pocos segundos en el proceso.

**Nota:** Al hacer *logout* en una cuenta del dispositivo, se procederá a cerrar sesión de todas las sesiones que quedasen abiertas en este (realizando la copia de seguridad correspondiente en el proceso).

#### 5.1.2. Mensajería FCM

La mensajería FCM se ha utilizado para añadir a todos los usuarios de la aplicación y a sus correspondientes dispositivos a un tema llamado “all”. A través de este se pueden enviar mensajes de difusión que llegan a todos los usuarios.

Concretamente hay uno programado para las 11:00am y 8:00pm de cada día que recibirán todos los usuarios con la aplicación instalada, a modo de recordatorio de hacer uso de la APP. En caso de estar utilizando la aplicación en el momento no aparecerá, ya que la aplicación tiene que estar cerrada para que aparezca.

Los ficheros necesarios para que este servicio de mensajería de Firebase funcione son:

- **google-services.json:** Este fichero se enviará a parte del GitHub principal, puesto que no es seguro subirlo al contener la información del servicio de Firebase.
- **key.json:** Fichero con las credenciales necesarias del proyecto en Firebase, para hacer posible conectar la base de datos remota al servicio FCM y para que se puedan almacenar y gestionar

los tokens de los dispositivos desde el mismo servidor. Se encuentra en el siguiente *path* del servidor: `/home/maitane.firebaseio_credentials`.

- **notificacionFirebase.sh:** Este *script* no es estrictamente necesario para el funcionamiento de este servicio de mensajería, puesto que se pueden enviar notificaciones desde la consola de Firebase. Sin embargo, permite enviar una notificación a todos los usuarios recordándoles que usen la APP al ejecutarlo. La ejecución de este está programada para las 11am y 8pm mediante *crontab* haciendo que llegue la notificación a **todos los usuarios que no estén conectados a la aplicación en ese mismo momento** y se encuentra en el directorio del servidor: `/home/maitane` (también se enviará a parte).
- **MyFirebaseMessagingService.kt:** Este fichero se encuentra en la carpeta “Firebase” del proyecto y es donde viene la configuración necesaria para recibir la notificación enviada por la plataforma y gestionarla.

#### 5.1.3. Captar imágenes y guardarlas en el servidor

Las imágenes se han implementado en el perfil del usuario una vez hecho el inicio de sesión. Al darse la opción de capturar la imagen tanto por cámara como por galería, se ha decidido por esta última al tratarse simplemente de una foto de perfil (normalmente se añade desde la galería haciendo uso de una foto ya tomada).



**Figura 8:** Galería del dispositivo visualizada por PickMedia.

Para ello, se ha hecho uso de un PickMedia en el *MainActivity* que una vez seleccionada una foto de la galería, se encarga de transformarla al formato BitmapDrawable para poder subirla a la base de datos. Al cambiar la foto, esta se enviará al servidor comprimida y se guardará allí, para más tarde añadirla al perfil, por lo que esto puede tardar unos segundos.

#### 5.1.4. Uso correcto de la aplicación y pila de actividades

Para que la información no se pierda en la aplicación se ha hecho uso de los *ViewModel*, además de los *rememberSaveable* en las diferentes pantallas. El único caso en el que la información se borra es al girar la pantalla en el *Login*, puesto que es una pantalla diferente de las otras dos, pero si sucede una interrupción de la APP, los datos no se perderían.

En cuanto a las pilas de pantallas, se ha seguido el mismo proceso que la anterior entrega, volviendo a la pantalla principal antes de acceder a la siguiente, de forma que en la pila solo se almacenen la principal y la actual (al haber varios *NavHost* para las diferentes transiciones, puede llegar a haber hasta seis elementos en la pila).

### 5.2. Requisitos extra

#### 5.2.1. Content Provider

El proveedor se ha realizado haciendo uso del calendario y se encuentra en el fichero *ContentProvider.kt* de la carpeta “utils”. Mediante este, al crear un gasto futuro a hoy se añade un evento en el calendario a modo recordatorio de que ese gasto se va a producir (en caso de tener como preferencia hacer cambios en el calendario).

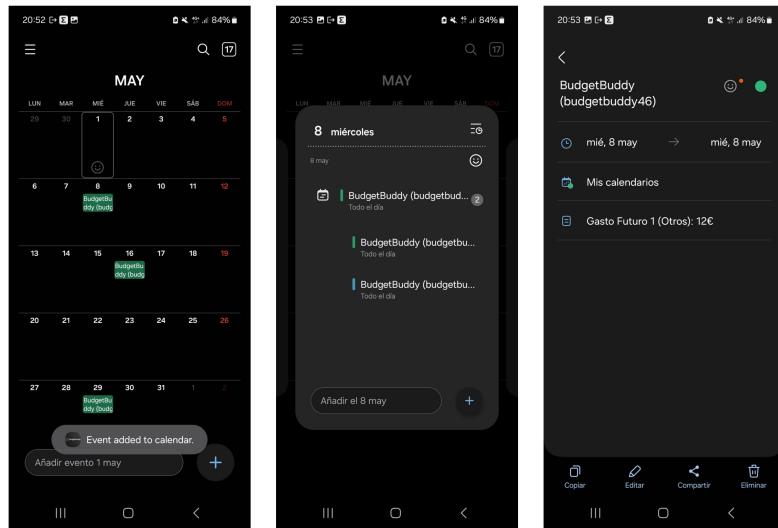


Figura 9: Eventos añadidos al calendario.

Este *provider* primero comprueba si existe un calendario de Google asociado a la cuenta de correo al usuario y se añadirá el evento en esta en caso de que exista. De no haber, comprobará si existe algún calendario local y en última instancia cogerá cualquier ID de calendario que exista en el dispositivo para añadir el evento. En caso de no tener calendarios, se avisaría al usuario mediante un mensaje *Toast*. Estos eventos se guardan de forma local, por lo que al cambiar de teléfono no aparecerían en el nuevo. Los eventos se complementan con las notificaciones de los *AlarmManager*.

## 5 REQUISITOS LLEVADOS A CABO

---

**Nota:** En los dispositivos que solo cuenten con Google Calendar (Pixel), debe haber al menos una cuenta añadida en el calendario Google para el correcto funcionamiento de este servicio.

### 5.2.2. Geolocalización

La geolocalización se ha utilizado en las pantallas para añadir y editar gastos, puesto que recogen la posición en la que se encuentra el usuario en ese momento y la añaden al gasto correspondiente. Una vez añadido, las coordenadas se guardarán como parte de los gastos.

El cliente que provee esta localización, al igual que el selector de imagen, se inicializa en el *MainActivity* y se usará siempre y cuando los permisos correspondientes estén activos.

### 5.2.3. Google Maps

Los mapas se utilizan en las pantallas de añadir, editar y en los *Card* de *Home* al abrirlos. El marcador señala la localización que hubiese guardada en el gasto con anterioridad o en el caso de la pantalla de añadir, la de la última lectura del cliente de geolocalización (el composable del mapa se encuentra en la carpeta “Shared” en *Comunes.kt*).

### 5.2.4. Widget

El *widget* está disponible en la carpeta “Widgets” del proyecto y se puede arrastrar al escritorio desde los *widgets* del teléfono. Es necesario una pantalla entera libre del teléfono para poder usarlo. Puesto que cuenta con una lista de los gastos y su información más relevante.

El *widget* se actualiza solo al iniciar sesión con un usuario nuevo o cerrar la sesión anterior, pero dispone de un botón de recarga ya que, en el entretanto, tarda unos 30 minutos en recargar por si solo (debido a limitaciones de la librería), por lo que si se quiere recargar antes, se puede forzar la actualización mediante este botón.

La pantalla del *widget* que especifica que hay que iniciar sesión siempre aparecerá en inglés (idioma por defecto de la aplicación). Al tener la sesión iniciada se mantendrá el idioma de preferencia del usuario.

### 5.2.5. Tarea programada mediante alarma

Estas alarmas complementan al proveedor de contenido mencionado anteriormente y se gestionan en la carpeta “AlarmManager” del proyecto. Tanto al añadir un gasto a futuro, como al hacer *login* con un usuario y para los gastos del día de hoy como los posteriores, se añadirá una alarma para el día que proceda a las 11am a forma de recordatorio de que ese gasto existe y se produciría el día en cuestión (en caso de los descargados para el día de hoy la alarma aparecerá unos segundos después de la descarga).

## REFERENCIAS

---

### Referencias

- [1] Microsoft Build. *Firebase Cloud Messaging*. URL: <https://learn.microsoft.com/es-es/xamarin/android/data-cloud/google-messaging/firebase-cloud-messaging>.
- [2] Google Cloud. *Google Cloud Engine*. URL: <https://cloud.google.com/products/compute>.
- [3] Android Developers. *Alarm Manager*. URL: <https://developer.android.com/reference/android/app/AlarmManager>.
- [4] Android Developers. *Cómo crear un widget de la app con Glance*. URL: <https://developer.android.com/develop/ui/compose/glance/create-app-widget?hl=es-419>.
- [5] Android Developers. *Cómo obtener la última ubicación conocida*. URL: <https://developer.android.com/develop/sensors-and-location/location/retrieve-current?hl=es-419>.
- [6] Android Developers. *Descripción general del proveedor de calendario*. URL: <https://developer.android.com/guide/topics/providers/calendar-provider?hl=es-419>.
- [7] Android Developers. *Selector de fotos*. URL: <https://developer.android.com/training/data-storage/shared/photopicker?hl=es-419>.
- [8] FastAPI. *SQL (Relational) Databases*. URL: <https://fastapi.tiangolo.com/tutorial/sql-databases/>.
- [9] Firebase. *Envía mensajes a varios dispositivos*. URL: <https://firebase.google.com/docs/cloud-messaging/android/send-multiple?hl=es-419>.
- [10] Jose Luis GS. *Crea tu API REST reactiva con Kotlin y Ktor Parte I*. URL: <https://joseluisgs.dev/blogs/2023/2023-05-29-reactive-api-rest-pt-i.html#creando-un-nuevo-proyecto-con-kotlin-y-ktor>.
- [11] Google Maps Platform. *Biblioteca de Maps Compose*. URL: <https://developers.google.com/maps/documentation/android-sdk/maps-compose?hl=es-419>.