



CASHFLOW

Jiacheng Lin Chen

Desarrollo de Aplicaciones Multiplataforma

Centro Integrado María Ana Sanz

ÍNDICE

1. Objetivos	3
2. Recursos	4
3. Enumeración y desarrollo de las fases	7
3.1. Análisis	7
3.2. Planificación	10
3.3. Diseño	12
3.3.1. Pantalla de registro y carga	12
3.3.2. Encabezado con barra de navegación y pantalla de gráficas	13
3.3.3. Pantalla de movimientos y pantalla para edición de un movimiento	14
3.3.4. Pantalla de configuración y edición de usuario	15
3.4. Desarrollo	16
1. App icon	16
2. App	17
3. AppShell	18
4. Verificación y animación en RegisterScreen	19
5. Base de datos con SQLite	21
6. Registrar usuario	24
7. Encriptación RSA	27
8. Transición RegisterScreen-LoadingScreen-GraphicsScreen	30
9. GraphicsScreen	33
10. ActivitiesScreen	45
11. ActivityEditScreen	51
12. ConfigurationScreen	55
13. ConfigScreenEdit	58
3.5. Pruebas	61
4. Conclusiones	66
5. Bibliografías y referencias	67

1. Objetivos

CashFlow es una aplicación desarrollada para la plataforma Android dedicada a la gestión económica del capital de un usuario con funcionalidades muy sencillas.

El principal objetivo de CashFlow es otorgar utilidad y comodidad al usuario en cuanto a la gestión de su capital. Así el usuario podrá tener un seguimiento diario de su capital, de todos los movimientos que va haciendo y también le obliga a estar pendiente diariamente con la aplicación, para que no se le pase nada por alto.

La aplicación busca que al usuario le resulte fácil realizar un seguimiento diario de su capital, que no le parezca muy laborioso o lento y que a la hora de mostrar la situación actual del dinero sea todo claro y conciso. No necesita estar conectado a ninguna cuenta del banco, el usuario simplemente introduce cualquier capital inicial y luego realiza los movimientos que vaya haciendo.

Además, la interfaz será lo más sencilla e intuitiva posible para que el usuario pueda interactuar con Cashflow de forma fluida y sea apta para usuarios de cualquier edad, por eso la aplicación también dispone de un uso muy sencillo.

CashFlow dispondrá de diversos apartados para el usuario. Una pantalla de gráficas donde se le mostrará al usuario los gastos e inversiones que se han realizado mediante gráficas, para que sea todo más visual y fácil de comprender. Otro apartado para introducir gastos e inversiones y ver todos los movimientos y una última pantalla de configuración para el usuario.

En términos generales, se ha desarrollado la aplicación CashFlow con finalidad de que los usuarios que lo adquieran tengan un mayor control del capital que ellos introduzcan y visualizar los movimientos que se van realizando. La aplicación estará al alcance para usuarios de cualquier edad, ya que el dinero es algo esencial y hay que gestionarlo de forma correcta, y CashFlow tiene como objetivo ayudar en mayor medida posible al usuario.

2. Recursos

Para el desarrollo de esta aplicación se ha necesitado un ordenador capaz de arrancar el software necesario. Por otro lado también se ha precisado de un teléfono móvil para iniciar y probar la aplicación, comprobando así que todo funciona correctamente. También un cable de tipo C a USB para conectar el dispositivo móvil al ordenador y así adquirir la aplicación en el móvil y trabajar desde ahí.

En cuanto a software, se ha necesitado el entorno de desarrollo Visual Studio 2022 con .Net Core 7 e instalado en el entorno la carga de trabajo “Desarrollo de la interfaz de usuario de aplicaciones multiplataforma de .NET”, donde se ha trabajado con .NET MAUI para el desarrollo. Y por último, una base de datos local de SQLite para guardar los datos del usuario y todos sus movimientos.

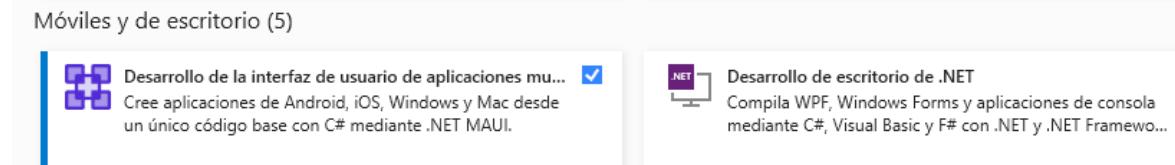
¿Por qué .NET MAUI?

MAUI (Multi-platform APP UI) es un macro multiplataforma dedicado a la creación de aplicaciones móviles y de escritorio nativas con C# y XAML. Es una plataforma de código abierto y es la evolución del conocido Xamarin Forms. Es debido a eso que es tan parecido, ya que se sigue trabajando con XAML y C#.

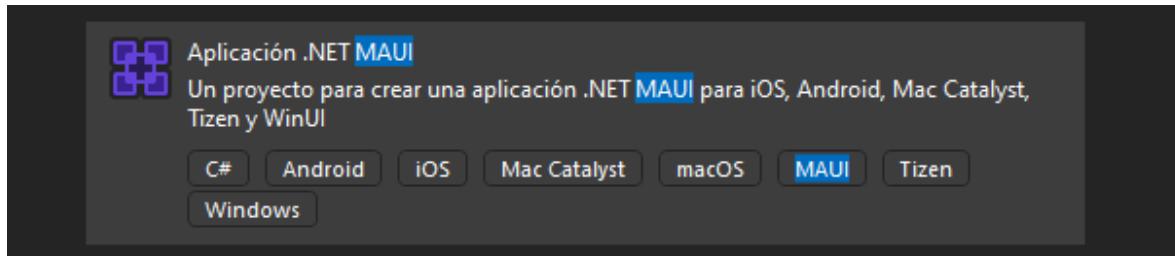
Pero MAUI, a parte de permitir también desarrollo de aplicaciones de escritorio, ofrece un mayor rendimiento y extensibilidad gracias a los controles de interfaz de usuario que están recopilados desde 0. Por lo general MAUI es una mejora en todos los aspectos respecto a Xamarin Forms, por eso se le conoce como el sucesor de Xamarin y está desarrollada con la misma tecnología de Xamarin, pero con algunas funciones diferentes para facilitar más el desarrollo.

En conclusión, las principales razones por las que se ha elegido .NET MAUI es por la mejora de rendimiento y sobre todo porque permite la personalización de la interfaz de forma más sencilla que en Xamarin Forms.

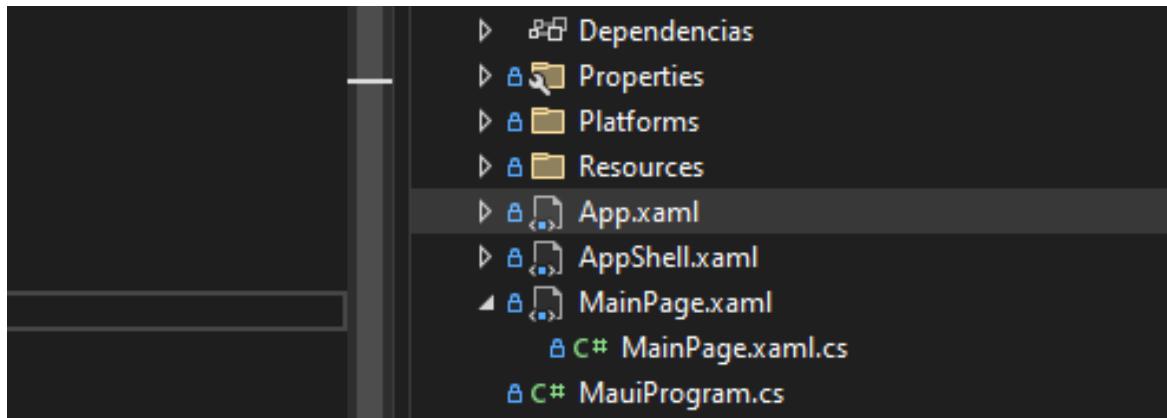
Para trabajar con .NET MAUI simplemente hay que instalar la siguiente carga de trabajo en el instalador de Visual Studio:



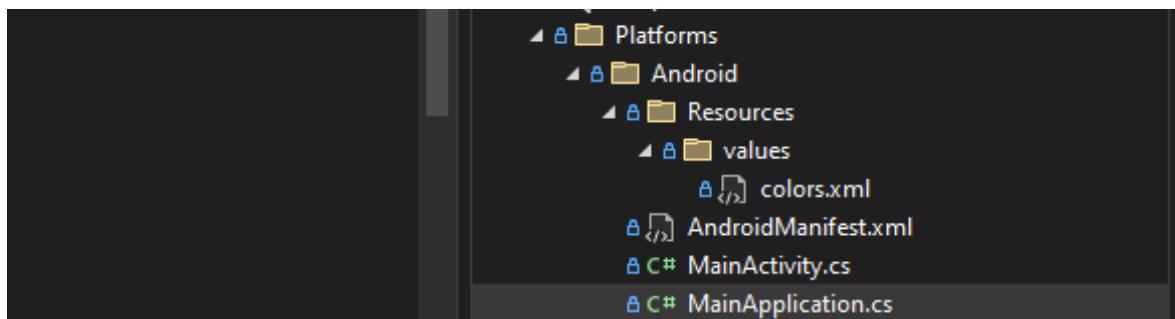
Una vez instalado ya permitirá crear un proyecto de MAUI en el Visual y trabajar con el framework:



Esto creará un nuevo proyecto de .NET MAUI con la siguiente estructura:



- En las Dependencias se encuentra lo necesario para cada una de las plataformas, net 7.0 para Android, MAC, iOS y Windows. Lo recomendable es no modificar nada en este apartado.
- Y en Properties se encuentra un archivo .json para determinadas configuraciones que afectan al proyecto.
- En la carpeta Platforms están las carpetas de cada plataforma donde se aloja información adicional de la plataforma, por ejemplo en Android se encuentra lo siguiente para que la aplicación se ejecute en Android:



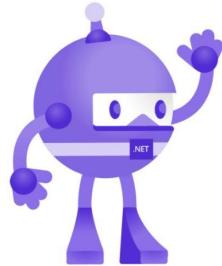
Es mejor no realizar ningún tipo de modificación, para evitar cualquier error.

- En la carpeta Resources se encuentran todos los recursos que serán comunes en todo el proyecto, lo necesario para la interfaz de la aplicación como fuentes, imágenes, estilos, etc.

- En el archivo App.xaml se configura los recursos que queramos que sean comunes en toda la app y en su clase (App.cs) se le asigna a MainPage la pantalla que abrirá la aplicación al ser ejecutada.
- AppShell.xaml define un tipo de apariencia de shell, donde se puede establecer un plantilla en el encabezado que será el mismo en toda la aplicación, será necesario este xaml para la barra de navegación.
- El archivo MainPage.xaml es un ContentPage donde se meten los elementos que se necesiten para realizar la interfaz.
- Y por último, la clase MauiProgram.cs que es la clase inicial donde se llevan ciertas configuraciones de la aplicación, por ejemplo las fuentes que se requieran.

Todos los archivos .xaml tienen su archivo .cs para meter código C# para ciertos eventos, la lógica de la aplicación, etc.

Si se ejecuta la aplicación inicial que crea MAUI se verá lo siguiente:



Una interfaz con un botón que muestra las veces que se ha pulsado el botón. Este diseño está definido en Mainpage.xaml.

Y a partir de este proyecto de partida se empieza a realizar los cambios requeridos para el desarrollo de la aplicación.

Hello, World!

Welcome to .NET Multi-platform App UI

Clicked 2 times

3. Enumeración y desarrollo de las fases

3.1. Análisis

Para la aplicación que se pretende hacer, es necesario comprobar si es algo viable y si es algo muy común. Existen muchas aplicaciones para el control de la capital, un ejemplo muy obvio es la de los propios bancos, pero estas apps suelen ser bastante complejas normalmente. Y a veces lo que se busca es la simplicidad, sobre todo para la gente de avanzada edad que no tiene mucho conocimiento en estos campos.

En cuanto a las funcionalidades, CashFlow es una aplicación sencilla e intuitiva, donde el usuario puede visualizar los movimientos que ha ido realizando durante el mes, el año y también una comparación entre ellas, todo esto mostrado en diferentes tipos de gráficas. También dispone de otro apartado para la introducción de cualquier gasto e inversión y se muestran todos estos movimientos abajo. Por último un apartado de configuración, donde se muestra la información del usuario y este podrá editarlo o borrarlo.

Y en cuanto a los recursos necesarios para el desarrollo de CashFlow, está ya especificado en el apartado anterior. Un software y un hardware y nada más.

La aplicación consta de varias pantallas y cada pantalla con sus clases, además de clases a partes que proporcionarán las funcionalidades necesarias. Un diagrama de todas las clases y con sus relaciones sería como la siguiente:

→ Pantallas de la aplicación:

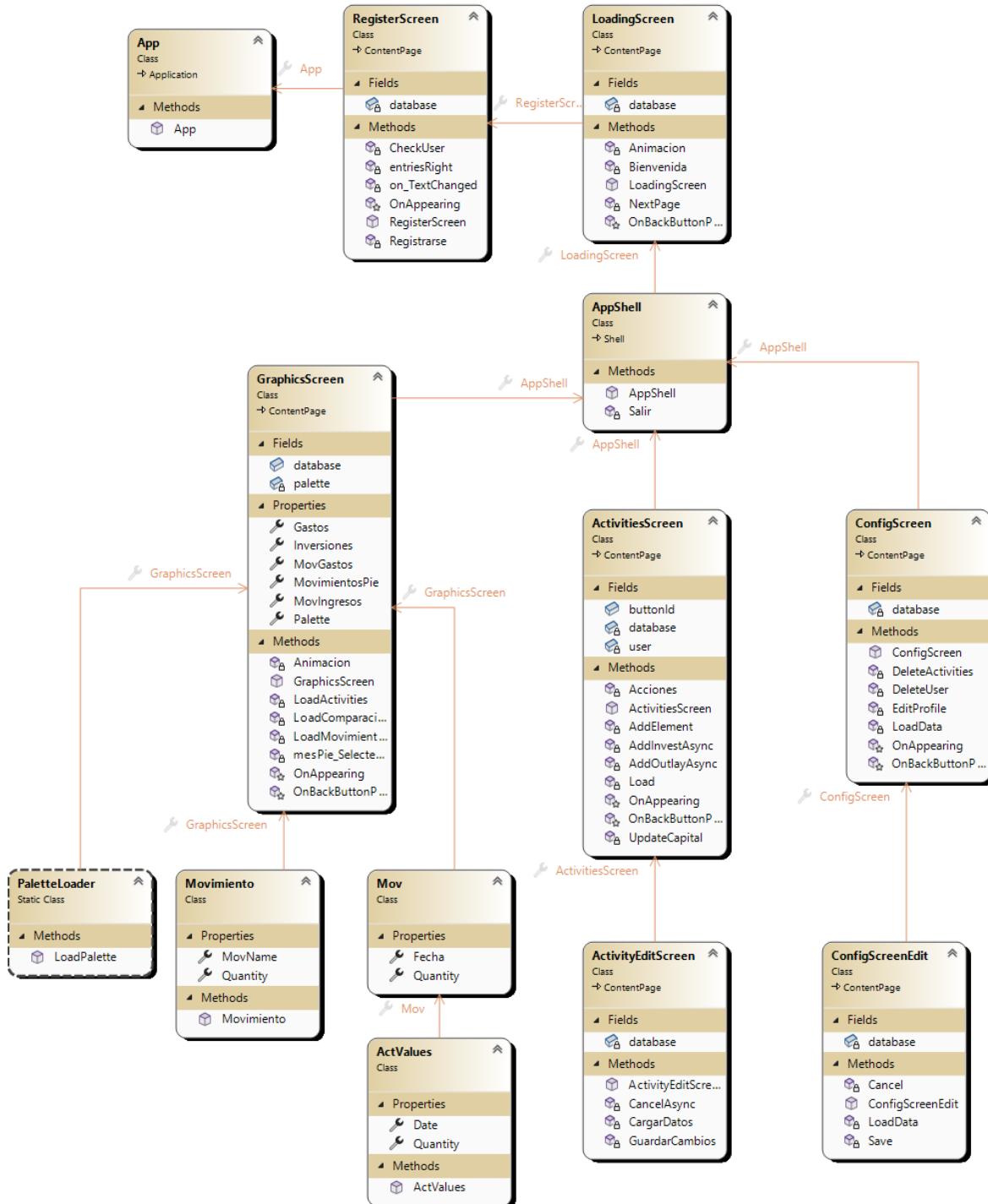


Diagrama de clases de las pantallas

→ Base de datos y clases que implementan interfaces.

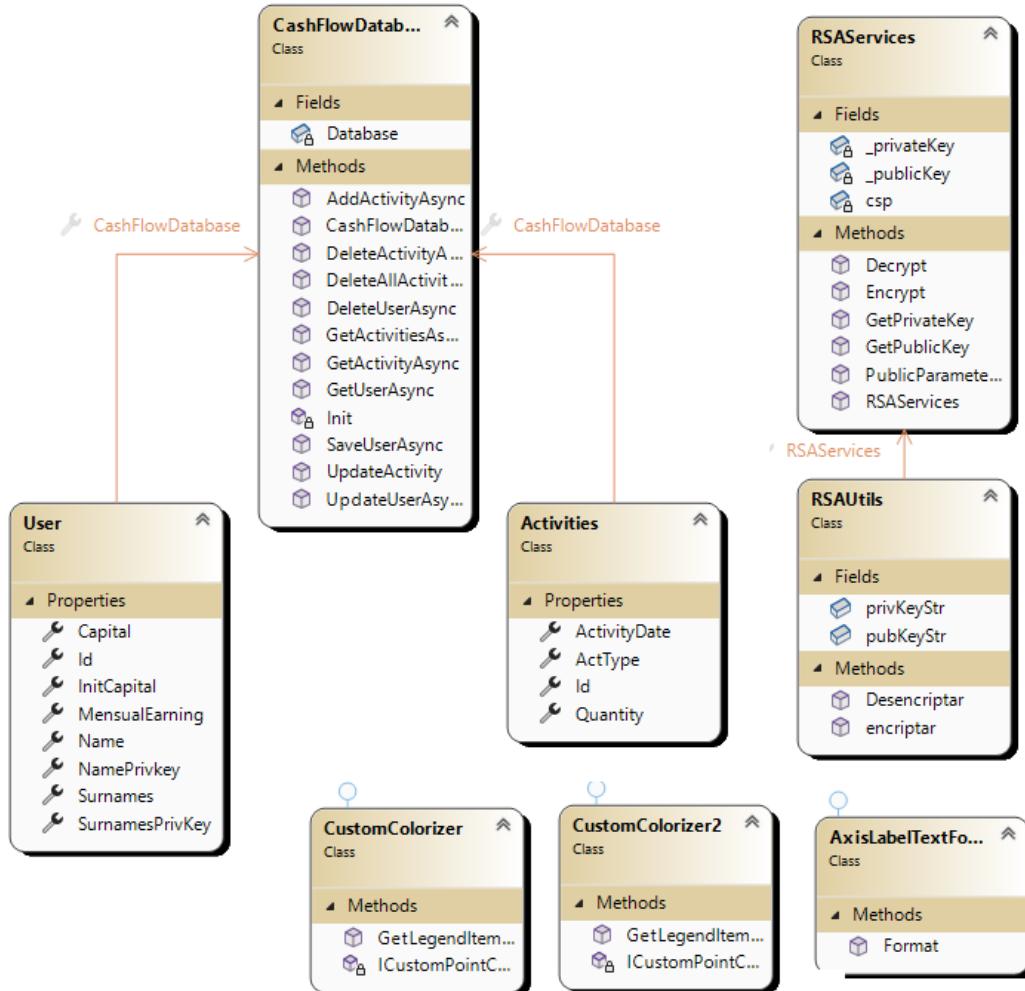


Diagrama de clases resto

Por último, un diagrama de secuencia para ver las líneas de vida o procesos que se originan de la interacción entre el usuario y la aplicación cuando este es abierto. Habría 3 capas: Usuario - GUI - BBDD, con esto se podrá visualizar la coexistencia entre estas tres capas.

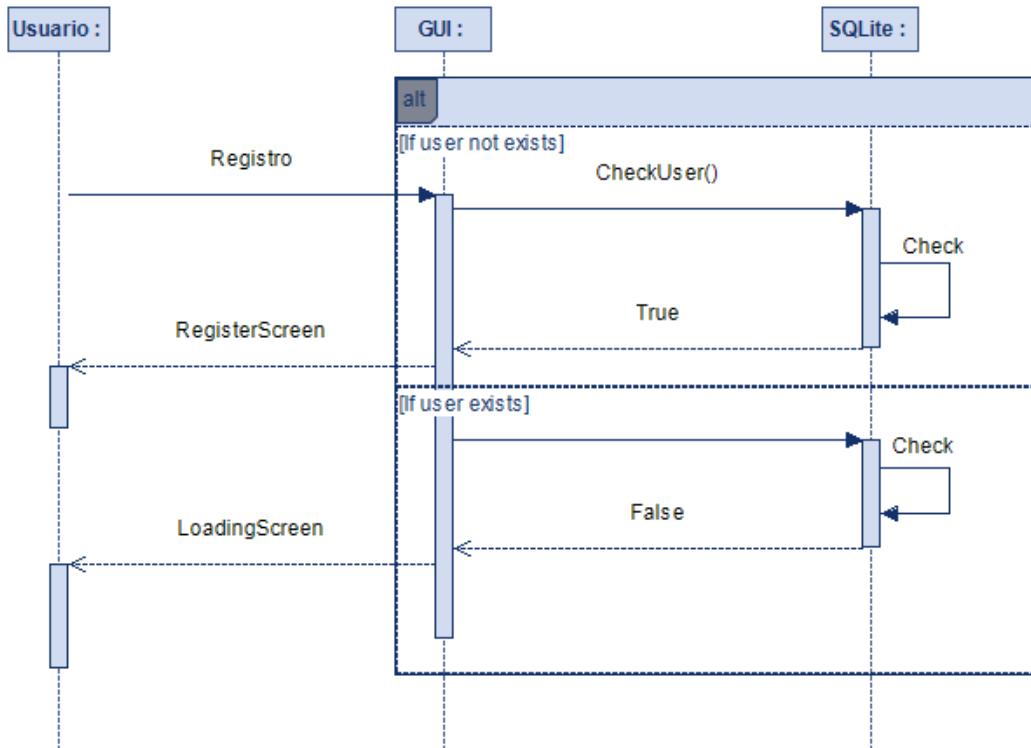


Diagrama de secuencia de Registro

3.2. Planificación

El proyecto de la aplicación se creará en un repositorio de GitHub para así realizar commits y guardar los cambios que se irán realizando, esto también permite trabajar en el proyecto en otros equipos si en algún momento es necesario. En cuanto al desarrollo de CashFlow, lo primero será realizar el diseño deseado de la aplicación (la fuente, los colores, gráficos, menús, etc.) y una vez acabado ir realizando de forma ordenada los distintos apartados que tendrá. Mientras se va realizando el diseño de los distintos apartados, se comenzará a trabajar en el código para establecer las funcionalidades y la lógica necesaria para cada apartado. Y todo esto determinará el funcionamiento de la aplicación.

Para trabajar con GitHub es necesario enlazarlo con el proyecto. Para ello, después de crear el proyecto con .NET MAUI, se va al menú Git y se selecciona Crear repositorio Git y aceptarlo, creando así un nuevo repositorio en la cuenta de GitHub que se ha especificado. Una vez hecho esto, en Visual se podrá realizar commits con los cambios que se vayan realizando en el proyecto, hacer pull o push y sincronizar el proyecto si se ha realizado cambios desde otro equipo.

The screenshot shows three main windows related to a GitHub repository named "CashFlow".

- Top Window:** A "Enviar cambios a un nuevo repositorio remoto" (Send changes to a new remote repository) dialog. It lists "GitHub" as the target, with "Default (VisualStudio)" selected as the template. Other options like ".gitignore" and "LICENSE" are available. The "Crear un nuevo repositorio de GitHub" (Create a new GitHub repository) section is also visible, showing "jlincdam1" as the account, "jlincdam1" as the owner, and "CashFlow" as the repository name. A checked checkbox indicates it's a private repository.
- Middle Window:** The GitHub repository page for "jlincdam1 / CashFlow". It shows 1 branch and 0 tags. Recent commits include:
 - "Agregar archivos de proyecto." by jlincdam1 (2 commits ago)
 - "Agregar archivos de proyecto." by jlincdam1 (1 minute ago)
 - "Agregar .gitattributes y .gitignore." by jlincdam1 (1 minute ago)
 - "Agregar .gitattributes y .gitignore." by jlincdam1 (1 minute ago)
 - "Agregar archivos de proyecto." by jlincdam1 (1 minute ago)
- Bottom Window:** A "Cambios de GIT: CashFlow" (Git Changes: CashFlow) dialog from Visual Studio. It shows the "main" branch with no staged changes. A message box asks "Escriba un mensaje <Requerido>". Below it are buttons for "Confirmar todo" (Commit all) and "Rectificar" (Revert). A status message at the bottom says "No hay cambios 'unstaged' en el directorio de trabajo."

3.3. Diseño

CashFlow tendrá un logo que llama bastante la atención para atraer a los clientes. La aplicación tendrá una interfaz gráfica intuitiva y con colores que sean cómodos para la vista, permitiendo así que el usuario pueda interactuar con ella de forma sencilla y cómoda. La aplicación tendrá una pantalla de registro y una vez registrado el usuario dispondrá de 3 diferentes apartados. El apartado de gráficas, para visualizar las inversiones y gastos que se ha ido realizando, el apartado de movimientos, donde el usuario introducirá los gastos e inversiones y, por último, un apartado de configuración para editar información del usuario. Todo con un encabezado para la navegación.

El diseño está realizado en Figma, es sólo uno inicial, puede haber cambios en el futuro:

3.3.1. Pantalla de registro y carga



3.3.2. Encabezado con barra de navegación y pantalla de gráficas



3.3.3. Pantalla de movimientos y pantalla para edición de un movimiento

MOVIMIENTOS

Inversión
Añada una inversión

+ Añadir

Gasto
Añada un gasto

+ Añadir

GASTOS E INVERSIONES:

- Inversión + 1000.00€ 01/04/2023
- Inversión + 105.80€ 31/03/2023
- Gasto - 851.25€ 25/03/2023

Editar movimiento

Tipo de movimiento ▾

✓ Inversión

Gasto

Cantidad (€)
1200

Fecha
08/17/2023

Cancelar Guardar

3.3.4. Pantalla de configuración y edición de usuario

The screenshot displays two screens of the CashFlow mobile application. The left screen is titled 'AJUSTES' (Settings) and shows 'Información del usuario' (User Information). It lists four items: 'Nombre: Jiacheng', 'Apellidos: Lin Chen', 'Capital: 5000€', and 'Ganancia mensual: 1500€'. Below this information is a button labeled 'Editar perfil' (Edit profile). The right screen is titled 'Editar perfil' (Edit profile) and contains three input fields: 'Nombre' (Name) with the value 'Carlos', 'Apellidos' (Last Name) with the value 'Larrondo', and 'Ganancia mensual (€)' (Monthly Profit) with the value '1600.95'. At the bottom of this screen are two buttons: 'Cancelar' (Cancel) and 'Guardar' (Save). Both screens feature a green header bar with a back arrow icon and a circular logo containing a dollar sign and a gear.

Nombre:	Jiacheng
Apellidos:	Lin Chen
Capital:	5000€
Ganancia mensual:	1500€

Nombre: Carlos

Apellidos: Larrondo

Ganancia mensual (€): 1600.95

Cancelar Guardar

Editar perfil

3.4. Desarrollo

Para el desarrollo de la aplicación, se realizará en Visual Studio 2022 y todos los cambios se verán reflejados en un repositorio de GitHub. Primero se realizará la interfaz de la aplicación, pero a la par introduciendo también el código C# necesario para su funcionamiento.

En primer lugar, se introducirá todos los recursos necesarios para CashFlow en la carpeta Resources. En AppIcon el icono de la aplicación, en Fonts se añadirá una nueva fuente para toda la app “Montserrat-Medium”, en Images el icono de CashFlow junto a los tres imágenes para el apartado de navegación y otra imagen para el botón de salida. Y por último en Splash el fondo de registro de la aplicación. Todos los recursos deben tener un nombre lógico para poder llamarlos cuando sea necesario su uso.

1. App icon

Para establecer el icono de la aplicación hay que modificar el AndroidManifest.xml y el archivo del proyecto CashFlow.csproj.

```
<encoding="utf-8"?>
<![CDATA[
<!-- App Icon -->
<item android:icon="@mipmap/cashflowicon" android:roundIcon="@mipmap/cashflowicon_round" android:label="AppIcon" />
<item android:name="android.permission.ACCESS_NETWORK_STATE" />
<item android:name="android.permission.INTERNET" />
]]>
```

Se cambia android:icon y android:roundIcon después del @minimap/ por el nombre del archivo de imagen que se encuentra en AppIcon.

```
<ItemGroup>
    <!-- App Icon -->
    <MauiIcon Include="Resources\AppIcon\cashflowicon.png"/>

    <!-- Splash Screen -->
    <MauiSplashScreen Include="Resources\Splash\splash.svg" Color="#68DF8C" BaseSize="128,128" />

    <!-- Images -->
    <MauiImage Include="Resources\Images\*" />
    <MauiImage Update="Resources\Images\dotnet_bot.svg" BaseSize="168,208" />
```

Se modifica el elemento MauiIcon la propiedad Include por la ruta del nuevo ícono. También se ha modificado el Color de MauiSplashScreen por otro más verde para conjuntar mejor con la aplicación.

2. App

Este ítem, viene creado con el proyecto, como ya se ha explicado anteriormente es el que inicializa la pantalla que se ejecutará al abrir la aplicación. Y en su xaml define los recursos para la aplicación, por defecto coge Colors.xaml y Styles.xaml de la carpeta Resources/Styles/. El único cambio que se va a realizar es en código C#, donde asignaremos la pantalla que deberá abrir la aplicación y también que sea un NavigationPage para poder ir cambiando de pantalla cuando sea necesario.

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:CashFlow"
    xmlns:android="clr-namespace:Microsoft.Maui.Controls.PlatformConfiguration.AndroidSpecific;assembly=Microsoft.Maui.Controls.PlatformConfiguration.AndroidSpecific"
    android:Application.WindowSoftInputModeAdjust="Resize"
    x:Class="CashFlow.App">
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="Resources/Styles/Colors.xaml" />
                <ResourceDictionary Source="Resources/Styles/Styles.xaml" />
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

```
namespace CashFlow;

5 references
public partial class App : Application
{
    0 references
    public App()
    {
        InitializeComponent();
        MainPage = new NavigationPage(new RegisterScreen());
    }
}
```

Para el desarrollo de las diferentes pantallas de CashFlow, se guardarán todos en la carpeta PhoneScreens (incluido AppShell) para que esté todo agrupado en un solo directorio.

3. AppShell

AppShell es un elemento que también viene creado en el proyecto (con su diseño XAML y su clase), es el que muestra la barra de navegación. En esta aplicación va a haber un encabezado con un botón de imagen para salir de la aplicación, el título del apartado en el que se encuentra el usuario y el ícono de CashFlow. Y debajo una barra de navegación para los tres apartados que habrá. Para esto se usará el elemento TabBar y dentro tres Tab para cada apartado, definiendo las propiedades de Icon y Title. Dentro de cada Tab se establecerá ShellContent con la propiedad ContentTemplate que asigna el contenido de ese apartado y para asignar elementos al encabezado, se define dentro el Shell.TitleView. Se define un StackLayout con dentro los elementos de ImageButton, una Label para el título y un Image para el ícono de la aplicación. Es importante también desactivar el Shell.FlyoutBehavior para que no sea un menú desplegable.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Shell
    x:Class="CashFlow.PhoneScreens.AppShell"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:CashFlow.PhoneScreens"
    Shell.FlyoutBehavior="Disabled" BackgroundColor="#9FC0AF" Shell.TabBarBackgroundColor="#ADCAC4">

    <TabBar>
        <Tab Icon="graphics.png" Title="Gráficas">
            <ShellContent ContentTemplate="{DataTemplate local:GraphicsScreen}">
                <Shell.TitleView>
                    <StackLayout Orientation="Horizontal" Margin="0, 0, 15, 0">
                        <ImageButton Source="leave.png" WidthRequest="35" HorizontalOptions="Start"/>
                        <Label Text="GRÁFICAS" TextColor="Black" FontSize="20" FontFamily="Montserrat-Medium" />
                        <Image Source="cashflowicon.png" WidthRequest="55" HorizontalOptions="End"/>
                    </StackLayout>
                </Shell.TitleView>
            </ShellContent>
        </Tab>
        <Tab Icon="manage.png" Title="Gestión">
            <ShellContent ContentTemplate="{DataTemplate local:ActivitiesScreen}">
                <Shell.TitleView>
                    <StackLayout Orientation="Horizontal" Margin="0, 0, 15, 0">
                        <ImageButton Source="leave.png" WidthRequest="35" HorizontalOptions="Start"/>
                        <Label Text="GASTOS E INVERSIONES" TextColor="Black" FontSize="20" FontFamily="Montserrat-Medium" />
                        <Image Source="cashflowicon.png" WidthRequest="55" HorizontalOptions="End"/>
                    </StackLayout>
                </Shell.TitleView>
            </ShellContent>
        </Tab>
        <Tab Icon="settings.png" Title="Ajustes">
            <ShellContent ContentTemplate="{DataTemplate local:ConfigScreen}">
                <Shell.TitleView>
                    <StackLayout Orientation="Horizontal" Margin="0, 0, 15, 0">
                        <ImageButton Source="leave.png" WidthRequest="35" HorizontalOptions="Start"/>
                        <Label Text="AJUSTES" TextColor="Black" FontSize="20" FontFamily="Montserrat-Medium" />
                        <Image Source="cashflowicon.png" WidthRequest="55" HorizontalOptions="End"/>
                    </StackLayout>
                </Shell.TitleView>
            </ShellContent>
        </Tab>
    </TabBar>
</Shell>
```

Por otro lado el ImageButton tendrá el método Salir() para el evento Clicked del botón, que simplemente cerrará la aplicación.

```
0 referencias
private async void Salir(object sender, EventArgs e)
{
    bool respuesta = await DisplayAlert("Salir", "¿Desea salir de CashFlow?", "Si", "No");
    if (respuesta)
    {
        Application.Current.Quit();
    }
}
```

4. Verificación y animación en RegisterScreen

Pantalla de registro en XAML, con un fondo que se ha establecido mediante una imagen y un logo junto a su título. También con Entries con sus Placeholders para que el usuario introduzca los datos necesarios, junto a labels que informan al usuario y el botón para registrarse:

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="CashFlow.PhoneScreens.RegisterScreen" BackgroundImageSource="fondo_registro.jpg">

    <ScrollView>
        <StackLayout Margin="30, 0, 30, 20">
            <Image Source="cashflowicon.png" WidthRequest="220" Margin="0, 20, 0, 15"/>

            <Label FontFamily="Montserrat-Medium" FontSize="22"
                FontAttributes="Bold"
                Text="BIENVENIDO A CASHFLOW!"
                HorizontalOptions="Center"/>

            <Label FontFamily="Montserrat-Medium" FontSize="16"
                HorizontalTextAlignment="Center"
                Margin="10, 40, 10, 15"
                Text="Regístrate para comenzar a gestionar su capital"/>

            <Entry x:Name="nombre"
                Margin="10, 10, 10, 5" Keyboard="Text"
                MaxLength="80" FontFamily="Montserrat-Medium"
                Placeholder="Nombre*" FontSize="16"
                PlaceholderColor="DarkSlateGray"
                TextChanged="on_TextChanged"/>

            <Entry x:Name="apellidos"
                Margin="10, 5, 10, 5" Keyboard="Text"
                MaxLength="100" FontFamily="Montserrat-Medium"
                Placeholder="Apellidos*" FontSize="16"
                PlaceholderColor="DarkSlateGray"
                TextChanged="on_TextChanged"/>

            <Entry x:Name="capitalI"
                Margin="10, 5, 10, 5"
                MaxLength="12" FontFamily="Montserrat-Medium"
                Keyboard="Numeric"
                Placeholder="Capital inicial*" FontSize="16"
                PlaceholderColor="DarkSlateGray"
                TextChanged="on_TextChanged"/>
        </StackLayout>
    </ScrollView>
</ContentPage>
```

```

<Entry x:Name="gananciaM"
       Margin="10, 5, 10, 5"
       MaxLength="12" FontFamily="Montserrat-Medium"
       Keyboard="Numeric"
       Placeholder="Ganancia mensual" FontSize="16"
       PlaceholderColor="DarkSlateGray"
       TextChanged="on_TextChanged"/>

<Button x:Name="btnRegistro"
        Text="REGISTRARSE" FontFamily="Montserrat-Medium"
        FontAttributes="Bold" FontSize="17"
        TextColor="Black" BackgroundColor="White"
        Margin="25, 30, 25, 5" CornerRadius="45"
        Clicked="Registrarse" IsEnabled="False" Opacity="0.2"/>

<Label FontFamily="Montserrat-Medium"
       FontSize="11"
       Margin="10, 25, 10, 5"
       TextColor="#000000"
       Text="Una vez registrado, la próxima vez que entre a la aplicación, accederá directamente al Inicio de CashF
<Button Text="eliminar" Clicked="Button_Clicked"/>
    </StackLayout>

```

Una vez se tenga la pantalla de registro se introducirá el código C# necesario para que el botón de Registrarse se active sólo si se han introducido los datos necesarios y son correctos, de lo contrario estará desactivado y con más opacidad. Para ello se emplea el método on_TextChanged para los entries. Por otra parte, si se pulsa el botón este realizará una pequeña animación para que el usuario vea que ya ha sido pulsado.

```

async void Registrarse(object sender, EventArgs e)
{
    await btnRegistro.ScaleTo(0.8, 50);
    await btnRegistro.ScaleTo(1, 50);
}

2 referencias
private bool entriesRight()
{
    return !string.IsNullOrWhiteSpace(nombre.Text) && !string.IsNullOrWhiteSpace(apellidos.Text) &&
           float.TryParse(capitalI.Text, out float result);
}

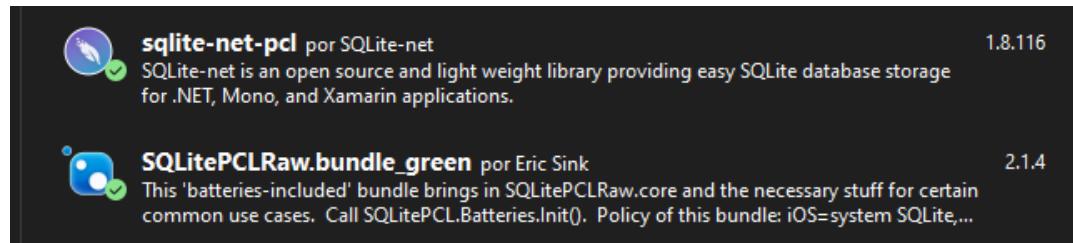
0 referencias
private void on_TextChanged(object sender, TextChangedEventArgs e)
{
    if (entriesRight() && string.IsNullOrWhiteSpace(gananciaM.Text))
    {
        btnRegistro.Opacity = 0.8;
        btnRegistro.IsEnabled = true;
    }
    else
    {
        if(entriesRight() && float.TryParse(gananciaM.Text, out float result))
        {
            btnRegistro.Opacity = 0.8;
            btnRegistro.IsEnabled = true;
        }
        else
        {
            btnRegistro.Opacity = 0.2;
            btnRegistro.IsEnabled = false;
        }
    }
}

```

5. Base de datos con SQLite

Cuando el usuario le de a registrarse, se guardarán todos los datos en una base de datos local y llevará al usuario a la siguiente pantalla que sería la de gráficas y mostrando la pantalla de carga de por medio.

Para la base de datos se usará SQLite, para ello primero se instalan los siguientes paquetes NuGet:



Una vez instalado hay que crear los modelos necesarios (se almacenarán en la carpeta Models), el del usuario y los movimientos que él mismo irá introduciendo a la aplicación:

```
namespace CashFlow.Models
{
    public class User
    {
        [PrimaryKey]
        public int Id { get; set; }
        public string Name { get; set; }
        public string Surnames { get; set; }
        public float InitCapital { get; set; }
        public float Capital { get; set; }
        public float MensualEarning { get; set; }
        public string NamePrivkey { get; set; }
        public string SurnamesPrivKey { get; set; }
    }
}
```

El modelo User es donde se encuentra toda la información del usuario que se guardará en la tabla. El id que siempre será el mismo ya que solo puede haber un usuario, su nombre y apellidos que se guardarán encriptados en RSA, el capital inicial, el capital actual que tiene y una ganancia mensual, esta cantidad se introducirá de forma automática el día 1 de cada mes como nuevo movimiento de inversión para la tabla de Activities. Por último las propiedades NamePrivkey y SurnamePrivKey para las claves privadas necesarias para desencriptar el nombre y los apellidos respectivamente.

```
namespace CashFlow.Models
{
    public class Activities
    {
        [PrimaryKey, AutoIncrement]
        public int Id { get; set; }
        public string ActType { get; set; }
        public float Quantity { get; set; }
        public DateTime ActivityDate { get; set; }
    }
}
```

En el modelo Activities se encuentran los datos de un movimiento, su Id que irá incrementando cada vez que se realicen nuevos movimientos, el tipo de movimiento (inversión o gasto), la cantidad de dinero que se ha invertido o gastado y la fecha del movimiento.

Para poder crear la base de datos y que sea almacenada, se necesitará establecer los constantes para la base de datos y esto se definirá en una clase Constants.cs, que se dejará en la raíz del proyecto por comodidad:

```
namespace CashFlow
{
    public static class Constants
    {
        public const string DatabaseFilename = "CashFlow.db3";

        public const SQLite.SQLiteOpenFlags Flags =
            // open the database in read/write mode
            SQLite.SQLiteOpenFlags.ReadWrite |
            // create the database if it doesn't exist
            SQLite.SQLiteOpenFlags.Create |
            // enable multi-threaded database access
            SQLite.SQLiteOpenFlags.SharedCache;

        public static string DatabasePath =>
            Path.Combine(FileSystem.AppDataDirectory, DatabaseFilename);
    }
}
```

Aquí se define el nombre de la base de datos, los flags para la base de datos y una vez definido todo se define la ruta que tendrá la base de datos en una variable que será usada más tarde.

En la clase CashFlowDatabase.cs, se creará la base de datos y las tablas necesarias, para crear la base de datos se usará SQLiteAsyncConnection para construirlo y usando como parámetro la ruta de la base de datos y los flags que fueron definidos en Constants.

```
4 referencias
public class CashFlowDatabase
{
    SQLiteAsyncConnection Database;
    1 referencia
    public CashFlowDatabase() { }

    8 referencias
    async Task Init()
    {
        if (Database is not null)
            return;

        Database = new SQLiteAsyncConnection(Constants.DatabasePath, Constants.Flags);
        await Database.CreateTableAsync<User>();
        await Database.CreateTableAsync<Activities>();
    }
}
```

Una vez completado el método Init(), ya se puede añadir los métodos necesarios para trabajar con las diferentes tablas, ya sea para añadir, eliminar, actualizar u obtener una fila de la tabla, eso sí se deberá llamar siempre al método Init() en cada método.

Métodos para tabla User:

```
1 reference
public async Task<User> GetUserAsync()
{
    await Init();
    return await Database.Table<User>().FirstOrDefaultAsync();
}

1 reference
public async Task<int> SaveUserAsync(User user)
{
    await Init();
    return await Database.InsertAsync(user);
}

1 reference
public async Task<int> DeleteUserAsync()
{
    await Init();
    return await Database.DeleteAllAsync<User>();
}

0 references
public async Task<int> UpdateUserAsync(User user)
{
    await Init();
    return await Database.UpdateAsync(user);
}
```

Métodos de Activities:

```
public async Task<List<Activities>> GetActivitiesAsync()
{
    await Init();
    return await Database.Table<Activities>().ToListAsync();
}
1 reference
public async Task<Activities> GetActivityAsync(int id)
{
    await Init();
    return await Database.Table<Activities>().Where(i => i.Id == id).FirstOrDefaultAsync();
}
2 references
public async Task<int> AddActivityAsync(Activities act)
{
    await Init();
    return await Database.InsertAsync(act);
}
1 reference
public async Task<int> DeleteActivityAsync(Activities act)
{
    await Init();
    return await Database.DeleteAsync(act);
}
1 reference
public async Task<int> DeleteAllActivities()
{
    await Init();
    return await Database.DeleteAllAsync<Activities>();
}
0 references
public async Task<int> UpdateActivityAsync(Activities act)
{
    await Init();
    return await Database.UpdateAsync(act);
}
```

6. Registrar usuario

Una vez implementada la base de datos, para registrar al usuario y añadirlo a la tabla User se llama al método SaveUserAsync de la base de datos y como solo va a haber un usuario no es necesario que el Id se autoincremente, se deja siempre en 1. Se crea un nuevo usuario de tipo User con la información de los Entries y se añade a la tabla, para trabajar con la base de datos se debe inicializarlo en el constructor y luego implementar todo en el método del botón Registrarse.

```
private readonly CashFlowDatabase database;
public string namePrivKey;
public string surnamesPrivKey;
1 reference
public RegisterScreen()
{
    InitializeComponent();
    NavigationPage.SetHasNavigationBar(this, false);
    database = new CashFlowDatabase();
}
```

```
User user;

double capI = Convert.ToDouble(capitalI.Text, CultureInfo.InvariantCulture);
string nombreEncriptado = RSAUtils.encriptar(nombre.Text.Trim());
string namePrivKey = RSAUtils.privKeyStr;
string apellidosEncriptado = RSAUtils.encriptar(apellidos.Text.Trim());
string surnamesPrivKey = RSAUtils.privKeyStr;

if (string.IsNullOrWhiteSpace(gananciaM.Text))
{
    user = new User
    {
        Id = 1,
        Name = nombreEncriptado,
        Surnames = apellidosEncriptado,
        InitCapital = (float)Math.Round(capI, 2),
        Capital = (float)Math.Round(capI, 2),
        NamePrivkey = namePrivKey,
        SurnamesPrivKey = surnamesPrivKey
    };
}
else
{
    double menE = Convert.ToDouble(gananciaM.Text, CultureInfo.InvariantCulture);
    user = new User
    {
        Id = 1,
        Name = nombreEncriptado,
        Surnames = apellidosEncriptado,
        InitCapital = (float)Math.Round(capI, 2),
        Capital = (float)Math.Round(capI, 2),
        MensualEarning = (float)Math.Round(menE, 2),
        NamePrivkey = namePrivKey,
        SurnamesPrivKey = surnamesPrivKey
    };
}

var response = await database.SaveUserAsync(user);
```

Además, se debe verificar si se ha introducido la ganancia mensual que es un campo no obligatorio, si es así también se coge el resultado del Entry para ese campo. Como se puede ver todos los campos de tipo float son guardados con dos décimas, si el usuario introduce un número con tres décimas este será redondeado.

Para verificar que se ha realizado el registro de forma correcta se le notificará al usuario si se ha completado el registro:

```
if(response > 0)
{
    await DisplayAlert("Éxito", "Usuario registrado correctamente", "Aceptar");
    await Task.WhenAll(
        this.FadeTo(0, 1000)
    );
    await Navigation.PushAsync(new LoadingScreen());
}
else
{
    await DisplayAlert("Error", "Error al añadir", "Aceptar");
}
```

Por último, el método CheckUser() sirve para revisar si existe ya un usuario en la base de datos, en caso de afirmativo se mandará al usuario directamente a la pantalla de carga para así acceder a la aplicación. En caso contrario se seguirá mostrando la pantalla de registro para que se registre ya que se trata de un usuario nuevo.

```
1 reference
private async void CheckUser()
{
    User user = await database.GetUserAsync();
    if (user != null)
    {
        await Task.WhenAll(
            this.FadeTo(0, 0)
        );
        await Navigation.PushAsync(new LoadingScreen());
    }
}

0 references
protected override void OnAppearing()
{
    CheckUser();
}
```

El método CheckUser es llamado en el método override de OnAppearing() para que se ejecute cada vez que se entra a la pantalla de registro.

7. Encriptación RSA

Como se puede observar en las capturas anteriores, para guardar los datos del usuario de forma segura, se encripta el nombre y los apellidos por RSA. Para ello se creará la clase RSAServices para definir las variables y los métodos necesarios para llevar a cabo esta encriptación.

```
9 references
public class RSAServices
{
    private RSACryptoServiceProvider csp;

    private RSAParameters _privateKey;
    private RSAParameters _publicKey;

    3 references
    public RSAServices()
    {
        csp = new RSACryptoServiceProvider(512);
        _privateKey = csp.ExportParameters(true);
        _publicKey = csp.ExportParameters(false);
    }

    1 reference
    public string GetPublicKey()
    {
        var sw = new StringWriter();
        var xs = new XmlSerializer(typeof(RSAParameters));
        xs.Serialize(sw, _publicKey);
        return sw.ToString();
    }

    1 reference
    public string GetPrivateKey()
    {
        var sw = new StringWriter();
        var xs = new XmlSerializer(typeof(RSAParameters));
        xs.Serialize(sw, _privateKey);
        return sw.ToString();
    }
}
```

Para encriptar en RSA se necesita una clave pública y para desencriptar la clave privada, es por ello que se necesita obtener estas dos claves. Por otro lado, para que la cadena de encriptación que va a ser guardada en la tabla User sea lo más corta posible, se define el tamaño de la cadena a 512 bits, que es el tamaño mínimo posible.

Los métodos GetPublicKey() y GetPrivateKey sirven para obtener las dos claves, para ello es necesario un StringWriter y un XmlSerializer y mediante el método Serialize de este último, usando como parámetros el StringWriter y la variable donde se almacena cada clave, se obtiene la clave en la variable. Y es devuelto como string.

Aparte de los métodos para obtener las dos claves, también es necesario un método para obtener los parámetros de cada clave, devolviéndolo en tipo RSAParameters y pide como parámetro la clave en string

```
2 references
public static RSAParameters PublicParametersFromXml(string data)
{
    XmlSerializer xml = new XmlSerializer(typeof(RSAParameters));
    object result;
    using (TextReader reader = new StringReader(data))
    {
        result = xml.Deserialize(reader);
    }
    return (RSAParameters)result;
}
```

Por último se establecen los métodos de Encrypt y Decrypt, donde se pide la cadena que se quiere encriptar y los RSAParameters de sus respectivos claves, devolviendo el texto encriptado en string.

```
1 reference
public string Encrypt(string plainText, RSAParameters publicKey)
{
    csp = new RSACryptoServiceProvider();
    csp.ImportParameters(publicKey);
    var data = Encoding.Unicode.GetBytes(plainText);
    var cypher = csp.Encrypt(data, false);
    return Convert.ToBase64String(cypher);
}

1 reference
public string Decrypt(string cypherText, RSAParameters privateKey)
{
    var dataBytes = Convert.FromBase64String(cypherText);
    csp = new RSACryptoServiceProvider();
    csp.ImportParameters(privateKey);
    var plainText = csp.Decrypt(dataBytes, false);
    return Encoding.Unicode.GetString(plainText); }
```

Para dar uso a estos dos métodos, se ha creado dos clases estáticas para en uno encriptar y en otro desencriptar junto a sus claves también estáticas en la clase RSAUtils.cs

```
5 references
public class RSAUtils
{
    public static string publicKeyStr;
    public static string privKeyStr;
2 references
    public static string encriptar(string str)
    {
        RSAServices rsa = new RSAServices();
        RSAServices rsa2 = new RSAServices();

        publicKeyStr = rsa.GetPublicKey();
        privKeyStr = rsa.GetPrivateKey();

        var publicKey = RSAServices.PublicParametersFromXml(publicKeyStr);

        string resulEncrit = rsa2.Encrypt(str, publicKey);

        return resulEncrit;
    }

1 reference
    public static string Desencriptar(string privkeyStr, string msjEncriptado)
    {
        RSAServices rsa = new RSAServices();

        var privKey = RSAServices.PublicParametersFromXml(privkeyStr);

        string resulDecrypt = rsa.Decrypt(msjEncriptado, privKey);

        return resulDecrypt;
    }
}
```

Como se puede observar, el método de encriptar no necesita como parámetro la clave pública y esto es debido a que dentro de este método se obtiene la clave tanto pública como privada y lo asigna a las variables estáticas. Por lo tanto, en encriptar() solo es necesario como parámetro el texto que se quiere encriptar, luego se obtienen los RSAParameters de la clave pública y se realiza la encriptación.

Por otro lado el método Desencriptar() sí requiere de un parámetro más que es la clave privada, obtiene mediante el método PublicParametersFromXml de RSAServices los parámetros de la clave privada y se lleva a cabo la desencriptación.

Y estos dos son los métodos usados para la encriptación del nombre y los apellidos, donde también se almacena las claves privadas de cada uno en la tabla, para así poder obtenerlos cuando se requiera desencriptar el nombre o los apellidos.

```
double capI = Convert.ToDouble(capitalI.Text, CultureInfo.InvariantCulture);
string nombreEncriptado = RSAUtils.encriptar(nombre.Text.Trim());
namePrivKey = RSAUtils.privKeyStr;
string apellidosEncriptado = RSAUtils.encriptar(apellidos.Text.Trim());
surnamesPrivKey = RSAUtils.privKeyStr;

if (string.IsNullOrWhiteSpace(gananciaM.Text))
{
    user = new User
    {
        Id = 1,
        Name = nombreEncriptado,
        Surnames = apellidosEncriptado,
        InitCapital = (float)Math.Round(capI, 2),
        Capital = (float)Math.Round(capI, 2),
        NamePrivateKey = namePrivKey,
        SurnamesPrivateKey = surnamesPrivKey
    };
}
```

8. Transición RegisterScreen-LoadingScreen-GraphicsScreen

Cuando el usuario se registre en CashFlow, se le llevará a una pantalla de carga y luego a la pantalla inicial, que en este caso es la pantalla de las gráficas. Esta navegación entre páginas es posible gracias a que en App() se inicializa a MainPage como un NavigationPage con RegisterScreen.

```
0 references
public App()
{
    InitializeComponent();

    MainPage = new NavigationPage(new RegisterScreen());
}
```

Esto permite lanzar al usuario a otra página mediante Navegación.PushAsync(). También si se quiere ocultar el NavigationBar en la pantalla de registro solo hay que establecerlo a false. En las siguientes capturas de pantalla se ve cómo ocultar la barra de navegación y cómo dirigir al usuario a otra pantalla. Durante la transición se simula una especie de animación donde la pantalla se desvanece gracias al método de Task WhenAll()

```
public RegisterScreen()
{
    InitializeComponent();
    NavigationPage.SetHasNavigationBar(this, false);
    database = new CashFlowDatabase();
}
```

```
await Task.WhenAll(
    this.FadeTo(0, 1000)
);
await Navigation.PushAsync(new LoadingScreen());
```

El LoadingScreen es simplemente una pantalla con una etiqueta que es rellenada en el código.

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="CashFlow.PhoneScreens.LoadingScreen"
    Title="LoadingScreen" BackgroundImageSource="fondo_registro.png">
    <ScrollView>
        <StackLayout Margin="30, 0, 30, 20">
            <Image Source="cashflowicon.png" WidthRequest="320" Margin="0, 150, 0, 0" HorizontalOptions="Center" />
            <Label x:Name="bienvenida" FontFamily="Montserrat-Medium" FontSize="36" FontAttributes="Bold" HorizontalTextAlignment="Center" Margin="0, 10, 0, 0" HorizontalOptions="Center"/>
        </StackLayout>
    </ScrollView>
</ContentPage>
```

En el código C#, se obtiene el usuario de la tabla User para conseguir su nombre y mostrarlo en la label de forma descriptada.

```
1 reference
private async void Bienvenida()
{
    User user = await database.GetUserAsync();
    string nombreDescriptado = RSAUtils.Desencriptar(user.NamePrivkey, user.Name);
    bienvenida.Text = "¡BIENVENIDO " + nombreDescriptado.ToUpper() + "!" ;
}
```

Para acompañar el efecto de desvanecimiento de la pantalla de registro y completar la animación, en el método Animacion se obtiene la pantalla LoadingScreen en 500 milisegundos, creando un efecto de aparición en la pantalla.

```
private async void Animacion()
{
    await Task.WhenAll(
        this.FadeTo(0, 0)
);
    await Task.WhenAll(
        this.FadeTo(1, 500)
);
```

Para que la pantalla no pase muy rápido, se mete un Task.Delay y antes de que se realice el Navigation.PushAsync, se desvanece la pantalla de carga otra vez para que se vea más fluido.

```
1 reference
private async void NextPage()
{
    await Task.Delay(2000);
    await Task.WhenAll(
        this.FadeTo(0, 1000)
    );
    await Navigation.PushAsync(new AppShell());
}
```

Y todo estos métodos son llamados en el constructor de la clase, también se inicializa la base de datos y se oculta la barra de navegación.

```
private readonly CashFlowDatabase database;
1 reference
public LoadingScreen()
{
    InitializeComponent();
    database = new CashFlowDatabase();
    NavigationPage.SetHasNavigationBar(this, false);
    Bienvenida();
    Animacion();
    NextPage();
}
```

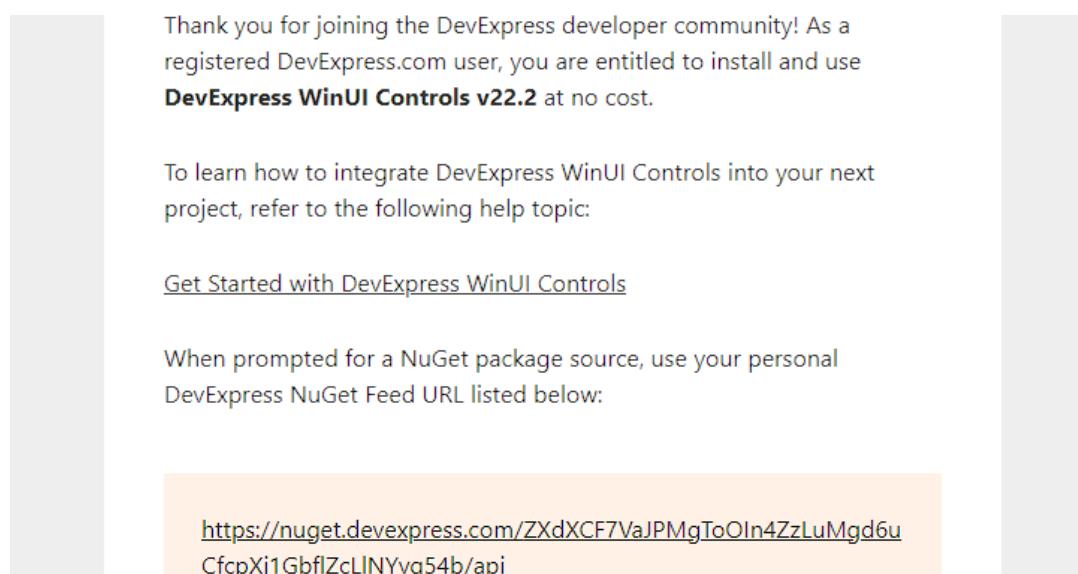
Por último, para bloquear la pantalla y el usuario no pueda usar el botón de retroceso se establece el método override OnBackButtonPressed()

```
0 references
protected override bool OnBackButtonPressed()
{
    return true;
}
```

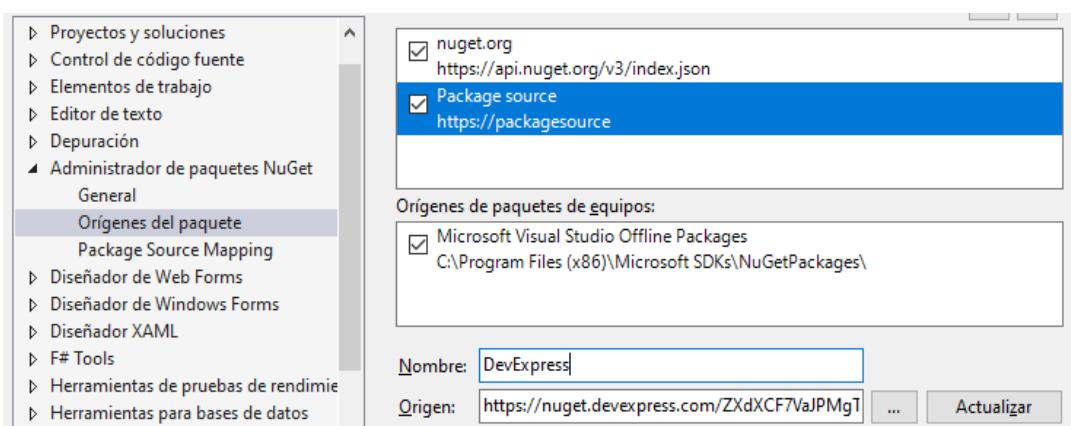
9. GraphicsScreen

En esta pantalla, se muestran gráficas que muestran al usuario cómo van las inversiones y gastos y la comparación entre estos. Por lo que antes de completar esta pantalla será mejor completar la pantalla de movimientos, para que así el usuario pueda introducir los movimientos y verlos reflejados en las gráficas.

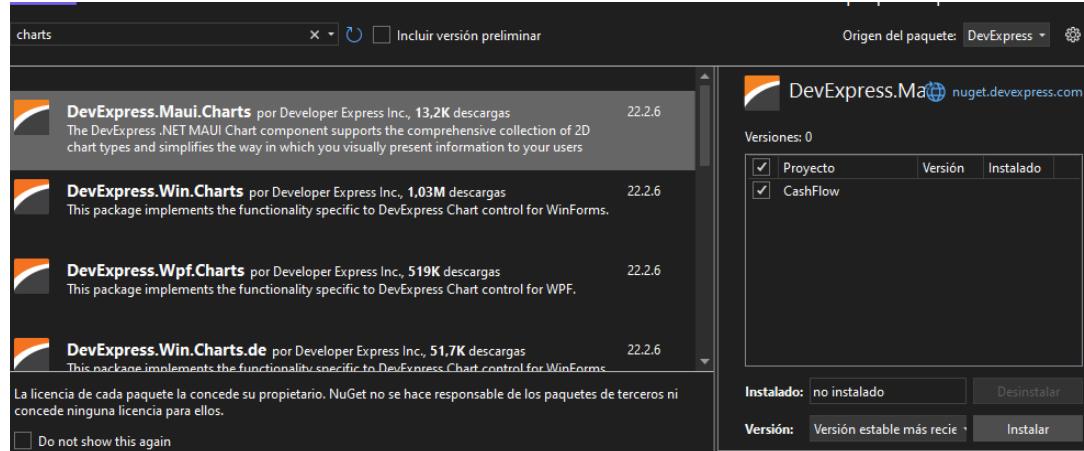
Para trabajar con gráficas, se ha usado unos paquetes NuGet de DevExpress. DevExpress es una página donde proporciona interfaces, formularios, etc. Y para darle uso a estos controles se necesita descargar los respectivos paquetes que proporciona DevExpress. Estos paquetes se consiguen registrándose en la página y aceptando la licencia, después de esto mandarán al correo que se ha usado para el registro una URL que servirá para introducir un nuevo origen de paquetes en Visual Studio:



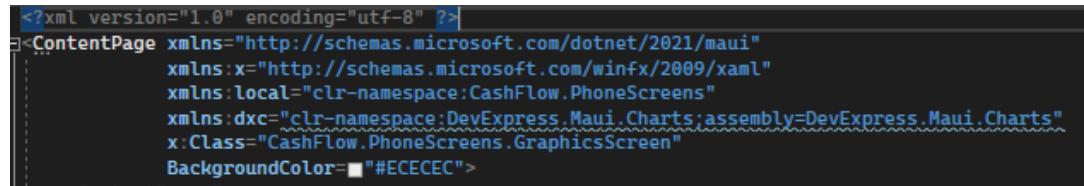
Esta URL se introduce como nuevo origen de paquetes en Herramientas/Opciones/Administrador de paquetes NuGet.



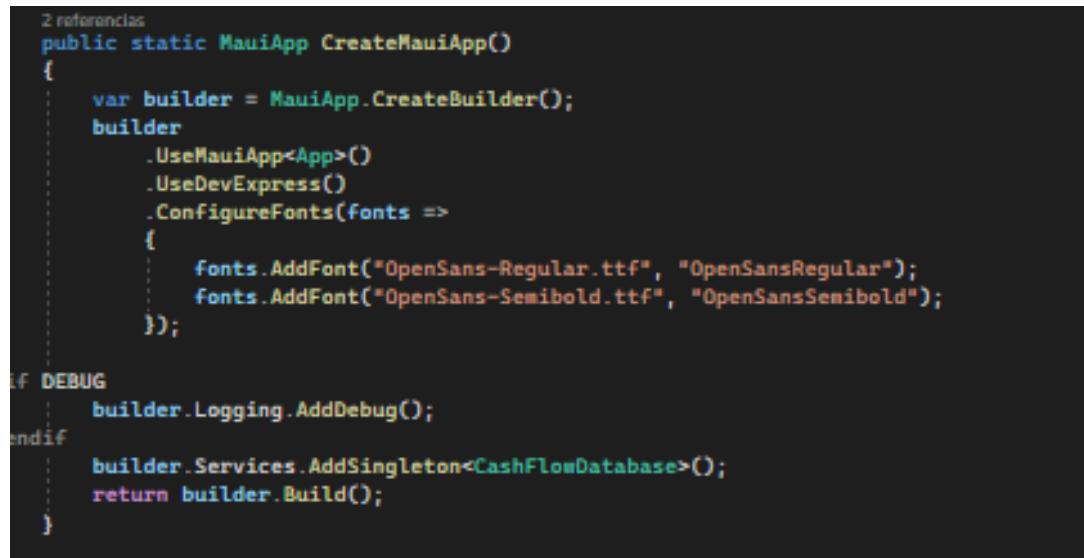
Una vez introducido el nuevo origen de paquetes NuGet se podrá agregar paquetes de DevExpress, en este caso para las gráficas se instalará el siguiente paquete:



Con este paquete se podrá trabajar con gráficas en el XAML, donde se establece el tipo de gráfico que se quiera (barras, tarta, etc.) y para poder darles uso habrá que añadir una nueva propiedad en ContentPage:



También es necesario llamar al servicio UseDevExpress() en MauiProgram.cs.



De la clase GraphicsScreen.cs se llamarán sus variables para rellenar las gráficas, donde se obtendrán los gastos e inversiones de la tabla Activities. Se van a emplear tres gráficas:

- PieChart: es la gráfica de sectores, donde se mostrarán los gastos e ingresos del mes actual del usuario para así ver la diferencia entre estos. Habrá un Picker encima de esta gráfica para elegir el mes del año actual en el que se quiere ver las inversiones y gastos.

Diseño en XAML:

```
<StackLayout Orientation="Horizontal" Margin="100, 20, 100, 0">
    <Label Text="Seleccione el mes: " FontFamily="Montserrat-Medium" HorizontalOptions="StartAndExpand" VerticalOptions="Center" />
    <Picker Title="Mes" x:Name="mesPie" FontFamily="Montserrat-Medium" HorizontalTextAlignment="Center" >
        <Picker.ItemsSource>
            <x:Array Type="{x:Type x:Int32}">
                <x:Int32>1</x:Int32>
                <x:Int32>2</x:Int32>
                <x:Int32>3</x:Int32>
                <x:Int32>4</x:Int32>
                <x:Int32>5</x:Int32>
                <x:Int32>6</x:Int32>
                <x:Int32>7</x:Int32>
                <x:Int32>8</x:Int32>
                <x:Int32>9</x:Int32>
                <x:Int32>10</x:Int32>
                <x:Int32>11</x:Int32>
                <x:Int32>12</x:Int32>
            </x:Array>
        </Picker.ItemsSource>
    </Picker>
</StackLayout>
```

```
<dxc:PieChartView SelectionBehavior="Hatch" HeightRequest="230" Margin="15, 10, 15, 10">
    <dxc:PieChartView.Series>
        <dxc:PieSeries>
            <dxc:PieSeries.Data>
                <dxc:PieSeriesDataAdapter x:Name="datosPie"
                    LabelDataMember="MovName"
                    ValueDataMember="Quantity"/>
            </dxc:PieSeries.Data>
            <dxc:PieSeries.HintOptions>
                <dxc:PieSeriesHintOptions PointTextPattern="{}{L}: {V$.##} €"/>
            </dxc:PieSeries.HintOptions>
        </dxc:PieSeries>
    </dxc:PieChartView.Series>
    <dxc:PieChartView.ChartStyle>
        <dxc:PieChartStyle x:Name="coloresPie" BackgroundColor="#ECECEC"/>
    </dxc:PieChartView.ChartStyle>
    <dxc:PieChartView.Hint>
        <dxc:PieHint Enabled="True"/>
    </dxc:PieChartView.Hint>
    <dxc:PieChartView.Legend>
        <dxc:Legend Orientation="TopToBottom"
            HorizontalPosition="RightOutside"
            VerticalPosition="Bottom">
            <dxc:Legend.Style>
                <dxc:LegendStyle BorderThickness="2" BorderColor="Gray"
                    MarkerSize="17" TextIndent="6" BackgroundColor="#ECECEC"
                    ItemsVerticalIndent="14" Padding="8">
                    <dxc:LegendStyle.TextStyle>
                        <dxc:TextStyle Size="12"/>
                    </dxc:LegendStyle.TextStyle>
                </dxc:LegendStyle>
            </dxc:Legend.Style>
        </dxc:Legend>
    </dxc:PieChartView.Legend>
</dxc:PieChartView>
```

Así se definen las gráficas en DevExpress, en este caso para la gráfica de secciones el PieChartView y dentro llegando a PieSeriesDataAdapter se introducirá las inversiones y gastos en el código. El LabelDataMember es el nombre del dato y ValueDataMember el valor. Si se quiere mostrar al usuario una etiqueta que muestre el tipo de movimiento y la cantidad total cuando este presione sobre una parte de la gráfica, con la opción PieSeriesHintOptions es posible con la propiedad PointTextPattern donde L es el LabelDataMember y V el ValueDataMember, el ValueDataMember está formateado para que se muestren dos decimales. También es necesario establecer el PieHint como true. Por último para que haya una leyenda simplemente se define PieChartView.Legend y dentro se define la posición y el estilo de la leyenda.

Código C#:

Para llenar la gráfica se usa una variable definida en la clase, en este caso una lista de tipo Movimiento llamado MovimientoPie.

```
public readonly CashFlowDatabase database;

public double gastos;
public double inversiones;
3 references
public List<Movimiento> MovimientosPie { get; set; }
```

En el método LoadActivities() se obtienen todas los movimientos de la base de datos y si no está vacía se suma a las variables gastos e inversiones dependiendo del tipo y se inicializa la variable MovimientosPie y se le asigna al DataSource de datosPie.

```
List<Activities> activities = await database.GetActivitiesAsync();
MovimientosPie = new List<Movimiento>()
{
    new Movimiento("Gastos", 0),
    new Movimiento("Inversiones", 0)
};
if (activities.Any())
{
    foreach (Activities activity in activities)
    {
        if(activity.ActivityDate.Month.ToString() == mesPie.SelectedItem.ToString() && activity.ActivityDa
    {
        if (activity.ActType == "Inversión")
        {
            MovimientosPie[1].Quantity += activity.Quantity;
        }
        else
        {
            MovimientosPie[0].Quantity += activity.Quantity;
        }
    }
    if(MovimientosPie[1].Quantity > 0 || MovimientosPie[0].Quantity > 0)
    {
        datosPie.DataSource = MovimientosPie;
        coloresPie.Palette = Palette;
    }
    else{
        datosPie.DataSource = null;
    }
}
else
```

La clase Movimiento contiene las siguientes dos propiedades, MovName para el nombre del movimiento y Quantity para la cantidad y estos dos son inicializados en el constructor que pide dos parámetros.

```
95  [
96
97     6 references
98  ↳public class Movimiento
99  {
100    1 reference
101    public string MovName { get; }
102    1 reference
103    public double Quantity { get; }
104
105    2 references
106    ↳public Movimiento(string actName, double quant)
107    {
108        this.MovName = actName;
109        this.Quantity = quant;
110    }
111 }
```

Por otro lado, para asignar los dos colores para gastos e inversiones respectivamente, se tendrá que asignar a la propiedad Palette del elemento PieChartStyle la variable de la clase Palette que es un array de tipo Color. A Palette se le asigna a su vez otra variable de solo lectura palette, que es inicializado en el constructo llamando al método estático LoadPalette de la clase estática PaletteLoader.

```
2
3     readonly Color[] palette;
4     1 reference
5     public Color[] Palette => palette;
6
7
8
9     database = new CashFlowDatabase();
10    MovimientosPie = new List<Movimiento>();
11    palette = PaletteLoader.LoadPalette("#f45a4e", "#25a966");
12 }
```

```
1 reference
5 ↳public static class PaletteLoader
6  {
7    1 reference
8    ↳public static Color[] LoadPalette(params string[] values)
9    {
10      Color[] colors = new Color[values.Length];
11      for (int i = 0; i < values.Length; i++)
12        colors[i] = Color.FromArgb(values[i]);
13      return colors;
14    }
15 }
```

- Bar Series: para la gráfica de barras habrá dos, uno para los gastos y otro para las inversiones. Cada gráfica mostrará la cantidad de cada movimiento que se ha realizado en cada mes del año actual, donde si el usuario presiona sobre una barra se mostrará la cantidad total de gastos o inversiones que se han realizado ese mes.

Diseño en XAML:

```
<dxc:ChartView HeightRequest="200" Margin="0, 10, 0, 0">
    <dxc:ChartView.Series>
        <dxc:BarSeries PointColorizer="{local:CustomColorizer}">
            <dxc:BarSeries.Data>
                <dxc:SeriesDataAdapter x:Name="gastosSeries" ArgumentDataMember="Fecha">
                    <dxc:ValueDataMember Member="Quantity" Type="Value" />
                </dxc:SeriesDataAdapter>
            </dxc:BarSeries.Data>
            <dxc:BarSeries.HintOptions>
                <dxc:SeriesHintOptions PointTextPattern="{}{A$YYYY-MM}: {V$.##} €"/>
            </dxc:BarSeries.HintOptions>
        </dxc:BarSeries>
    </dxc:ChartView.Series>
    <dxc:ChartView.ChartStyle>
        <dxc:ChartStyle BackgroundColor="LightGray" Padding="10"/>
    </dxc:ChartView.ChartStyle>
    <dxc:ChartView.AxisX>
        <dxc:DateTimeAxisX Style="{StaticResource axisStyle}" MeasureUnit="Month" GridSpacing="1">
            <dxc:DateTimeAxisX.LabelTextFormatter>
                <local:AxisLabelTextFormatter/>
            </dxc:DateTimeAxisX.LabelTextFormatter>
        </dxc:DateTimeAxisX>
    </dxc:ChartView.AxisX>
    <dxc:ChartView.AxisY>
        <dxc:NumericAxisY Style="{StaticResource axisStyle}" />
    </dxc:ChartView.AxisY>
    <dxc:ChartView.Hint>
        <dxc:Hint Enabled="True"/>
    </dxc:ChartView.Hint>
</dxc:ChartView>
```

```
</StackLayout>
<dxc:ChartView HeightRequest="200" Margin="0, 10, 0, 0">
    <dxc:ChartView.Series>
        <dxc:BarSeries PointColorizer="{local:CustomColorizer2}">
            <dxc:BarSeries.Data>
                <dxc:SeriesDataAdapter x:Name="inversionesSeries" ArgumentDataMember="Fecha">
                    <dxc:ValueDataMember Member="Quantity" Type="Value" />
                </dxc:SeriesDataAdapter>
            </dxc:BarSeries.Data>
            <dxc:BarSeries.HintOptions>
                <dxc:SeriesHintOptions PointTextPattern="{}{A$YYYY-MM}: {V$.##} €"/>
            </dxc:BarSeries.HintOptions>
        </dxc:BarSeries>
    </dxc:ChartView.Series>
    <dxc:ChartView.ChartStyle>
        <dxc:ChartStyle BackgroundColor="LightGray" Padding="10"/>
    </dxc:ChartView.ChartStyle>
    <dxc:ChartView.AxisX>
        <dxc:DateTimeAxisX Style="{StaticResource axisStyle}" MeasureUnit="Month" GridSpacing="1">
            <dxc:DateTimeAxisX.LabelTextFormatter>
                <local:AxisLabelTextFormatter/>
            </dxc:DateTimeAxisX.LabelTextFormatter>
        </dxc:DateTimeAxisX>
    </dxc:ChartView.AxisX>
    <dxc:ChartView.AxisY>
        <dxc:NumericAxisY Style="{StaticResource axisStyle}" />
    </dxc:ChartView.AxisY>
    <dxc:ChartView.Hint>
        <dxc:Hint Enabled="True"/>
    </dxc:ChartView.Hint>
</dxc:ChartView>
```

En las gráficas de barras la definición es muy parecida al de secciones, solo que se asignan otras variables de la clase para llenar los datos. Pero aparte de eso, en este caso, se asignará el color de cada gráfica con otra clase llamada CustomColorizer.cs (para gastos) o CustomColorizer2.cs (para inversiones) que se encuentra en la misma carpeta que GraphicsScreen. Por otro lado, en la gráfica de barras está la opción de establecer un estilo para los ejes X e Y mediante NumericAxisY o X dentro de ChartView.AxisY o X. En este caso se va a asignar un estilo que está definido en el propio XAML arriba del todo con nombre clave “AxisStyle”, donde se establece a negro las líneas de la gráfica. Se ha cambiado ya que por defecto vienen de un color grisáceo claro y no contrasta con el fondo gris. Por último, para mostrar los doce meses en el eje X simplemente se define la propiedad MeasureUnit como “Month”.

```
public class CustomColorizer : ICustomPointColorizer
{
    0 referencias
    Color ICustomPointColorizer.GetColor(ColoredPointInfo info)
    {
        return Color.FromArgb("#CC2626");
    }
    0 referencias
    public ILegendItemProvider GetLegendItemProvider()
    {
        return null;
    }
}
```

```
0 referencias
public class CustomColorizer2 : ICustomPointColorizer
{
    0 referencias
    Color ICustomPointColorizer.GetColor(ColoredPointInfo info)
    {
        return Color.FromArgb("#25a966");
    }
    0 referencias
    public ILegendItemProvider GetLegendItemProvider()
    {
        return null;
    }
}
```

```
    ShowInLabel="True"
    AxisLabelVisible="True"
    AxisLineVisible="True"/>
<dxc:AxisStyle x:Key="axisStyle" LineColor="Black" MajorGridlinesColor="Black"/>
</ContentPage.Resources>
<ContentPage>
```

Código C#:

```
3 referencias
public List<Mov> Gastos { get; set; }

3 referencias
public List<Mov> Inversiones { get; set; }
```

Estas son las dos variables que se usarán para llenar cada gráfica.

Y se llama al método privado LoadMovimientosSeries() para cargar los ingresos y gastos en cada gráfica. Se instancian las variables Gastos e Inversiones de tipo Mov como una lista con doce elementos, cada uno para un mes del año. Luego la cantidad de cada mes se modifica dependiendo de los movimientos que se han realizado y se comprueba también que el año en el que se ha realizado el movimiento es el mismo que el año actual.

```
private async void LoadMovimientosSeries()
{
    Gastos = new List<Mov>()
    {
        new Mov() { Fecha = new DateTime(2020, 1, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 2, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 3, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 4, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 5, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 6, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 7, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 8, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 9, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 10, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 11, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 12, 1), Quantity = 0 },
    };
    Inversiones = new List<Mov>()
    {
        new Mov() { Fecha = new DateTime(2020, 1, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 2, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 3, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 4, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 5, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 6, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 7, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 8, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 9, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 10, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 11, 1), Quantity = 0 },
        new Mov() { Fecha = new DateTime(2020, 12, 1), Quantity = 0 },
    };
}
```

```
    List<Activities> activities = await database.GetActivitiesAsync();
    if (activities.Count > 0)
    {
        foreach (Activities activity in activities)
        {
            if(activity.ActivityDate.Year == DateTime.Now.Year)
            {
                int mes = activity.ActivityDate.Month;
                if (activity.ActType == "Gasto")
                {
                    Gastos[mes - 1].Quantity += activity.Quantity;
                }
                else
                {
                    Inversiones[mes - 1].Quantity += activity.Quantity;
                }
            }
        }
        gastosSeries.DataSource = Gastos;
        inversionesSeries.DataSource = Inversiones;
    }
}
```

```
28 referencias
public class Mov
{
    24 referencias
    public DateTime Fecha { get; set; }
    26 referencias
    public double Quantity { get; set; }
}
```

- LineSeries: este último gráfico tendrá dos líneas, uno de gastos y otro de inversiones donde se mostrará la cantidad de cada uno mensualmente en el año actual. Así el usuario podrá ver la comparativa de cada mes de los movimientos que ha ido realizando durante el año. Cada línea se mostrará de un color y en cada intervalo de mes se podrá ver un punto, donde se puede pinchar y se mostrará la cantidad del movimiento que se ha realizado en ese mes. Con esta gráfica el usuario conseguirá sacar conclusiones de todos los movimientos que ha y está realizando y si debe de actuar de una forma más moderada.

Diseño XAML:

```

<dx:ChartView HeightRequest="350" Margin="0, 20, 0, 20">
    <dx:ChartView.Series>
        <dx:LineSeries x:Name="nombreIngresos"
            HintOptions="(StaticResource lineSeriesHintOptions)"
            MarkersVisible="True">
            <dx:LineSeries.Data>
                <dx:SeriesDataAdapter x:Name="ingresosDatos"
                    ArgumentDataMember="Date">
                    <dx:ValueDataMember Type="Value" Member="Quantity"/>
                </dx:SeriesDataAdapter>
            </dx:LineSeries.Data>
            <dx:LineSeries.Style>
                <dx:LineSeriesStyle Stroke="Green" StrokeThickness="2" MarkerSize="7">
                    <dx:LineSeriesStyle.MarkerStyle>
                        <dx:MarkerStyle Fill="Green"/>
                    </dx:LineSeriesStyle.MarkerStyle>
                </dx:LineSeriesStyle>
            </dx:LineSeries.Style>
        </dx:LineSeries>
        <dx:LineSeries x:Name="nombreGastos"
            HintOptions="(StaticResource lineSeriesHintOptions)"
            MarkersVisible="True">
            <dx:LineSeries.Data>
                <dx:SeriesDataAdapter x:Name="gastosDatos"
                    ArgumentDataMember="Date">
                    <dx:ValueDataMember Type="Value" Member="Quantity"/>
                </dx:SeriesDataAdapter>
            </dx:LineSeries.Data>
            <dx:LineSeries.Style>
                <dx:LineSeriesStyle Stroke="Red" StrokeThickness="2" MarkerSize="7">
                    <dx:LineSeriesStyle.MarkerStyle>
                        <dx:MarkerStyle Fill="Red"/>
                    </dx:LineSeriesStyle.MarkerStyle>
                </dx:LineSeriesStyle>
            </dx:LineSeries.Style>
        </dx:LineSeries>
    </dx:ChartView.Series>
    <dx:ChartView.AxisX>
        <dx:DateTimeAxisX Style="(StaticResource axisStyle)" MeasureUnit="Month" GridSpaci
            <dx:DateTimeAxisX.LabelTextFormatter>
                <local:AxisLabelTextFormatter/>
            </dx:DateTimeAxisX.LabelTextFormatter>
        </dx:DateTimeAxisX>
    </dx:ChartView.AxisX>
    <dx:ChartView.AxisY>

```

```

<dxc:ChartView>
    <dxc:ChartView.AxisY>
        <dxc:NumericAxisY Style="{StaticResource axisStyle}" />
    </dxc:ChartView.AxisY>
    <dxc:ChartView.Legend>
        <dxc:Legend VerticalPosition="BottomOutside"
            HorizontalPosition="Center"
            Orientation="LeftToRight">
            <dxc:Legend.Style>
                <dxc:LegendStyle BackgroundColor="LightGray">
                    <dxc:LegendStyle.TextStyle>
                        <dxc:TextStyle Size="12"/>
                    </dxc:LegendStyle.TextStyle>
                </dxc:LegendStyle>
            </dxc:Legend.Style>
        </dxc:Legend>
    </dxc:ChartView.Legend>
    <dxc:ChartView.Hint>
        <dxc:Hint>
            <dxc:Hint.Behavior>
                <dxc:TooltipHintBehavior ShowPointTooltip="True" ShowSeriesTooltip="False"/>
            </dxc:Hint.Behavior>
        </dxc:Hint>
    </dxc:ChartView.Hint>
    <dxc:ChartView.ChartStyle>
        <dxc:ChartStyle BackgroundColor="LightGray"/>
    </dxc:ChartView.ChartStyle>
</dxc:ChartView>

```

Para esta gráfica de comparación se usa el elemento LineSeries, donde se rellenan los datos de una manera parecida a las gráficas anteriores y para mostrar la etiqueta (HintOptions) se le asignará uno definido arriba en el propio XAML. En cuanto al estilo de la línea, se define con LineSeriesStyle, donde también se le añade un MarkerSize para que se vea el punto en cada eje del mes. A cada movimiento se le asigna un color diferente, para poder diferenciarlos y ver la comparación. Para mostrar la etiqueta en el Marker de la línea también habrá que asignar la propiedad ShowPointToolTip a true en TooltipHintBehavior y asignar ShowSeriesToolTip a false. Los estilos de los dos ejes también son asignados por el estilo definido arriba "axisStyle". Para finalizar, en esta gráfica se añadirá también una leyenda con su respectivo posicionamiento y estilo.

```

<ContentPage.Resources>
    <dxc:SeriesCrosshairOptions x:Key="lineSeriesHintOptions"
        PointTextPattern="{}{S}: {V$.##}€"
        ShowInLabel="True"
        AxisLabelVisible="True"
        AxisLineVisible="True"/>
    <dxc:AxisStyle x:Key="axisStyle" LineColor="Black" MajorGridlinesColor="Black"/>
</ContentPage.Resources>

```

Código C#:

Se usarán las variables MovIngresos y MovGastos de tipo Act, donde cargan a la gráfica en el método LoadComparacion().

```

4 referencias
public Act MovIngresos { get; set; }
4 referencias
public Act MovGastos { get; set; }

```

```

    i referencia
    private async void LoadComparacion()
    {
        MovIngresos = new Act(
            "Ingresos",
            new ActValues(new DateTime(0000, 1, 1), 0),
            new ActValues(new DateTime(0000, 2, 1), 0),
            new ActValues(new DateTime(0000, 3, 1), 0),
            new ActValues(new DateTime(0000, 4, 1), 0),
            new ActValues(new DateTime(0000, 5, 1), 0),
            new ActValues(new DateTime(0000, 6, 1), 0),
            new ActValues(new DateTime(0000, 7, 1), 0),
            new ActValues(new DateTime(0000, 8, 1), 0),
            new ActValues(new DateTime(0000, 9, 1), 0),
            new ActValues(new DateTime(0000, 10, 1), 0),
            new ActValues(new DateTime(0000, 11, 1), 0),
            new ActValues(new DateTime(0000, 12, 1), 0));

        MovGastos = new Act(
            "Gastos",
            new ActValues(new DateTime(0000, 1, 1), 0),
            new ActValues(new DateTime(0000, 2, 1), 0),
            new ActValues(new DateTime(0000, 3, 1), 0),
            new ActValues(new DateTime(0000, 4, 1), 0),
            new ActValues(new DateTime(0000, 5, 1), 0),
            new ActValues(new DateTime(0000, 6, 1), 0),
            new ActValues(new DateTime(0000, 7, 1), 0),
            new ActValues(new DateTime(0000, 8, 1), 0),
            new ActValues(new DateTime(0000, 9, 1), 0),
            new ActValues(new DateTime(0000, 10, 1), 0),
            new ActValues(new DateTime(0000, 11, 1), 0),
            new ActValues(new DateTime(0000, 12, 1), 0));
        List<Activities> activities = await database.GetActivitiesAsync();

        if (activities.Count > 0)
        {
            foreach (Activities activity in activities)
            {
                if (activity.ActivityDate.Year == DateTime.Now.Year)
                {
                    int mes = activity.ActivityDate.Month;
                    foreach (ActValues a in MovGastos.Values)
                    {
                        if (activity.ActType == "Gasto" && a.Date.Month == mes)
                        {
                            a.Quantity += activity.Quantity;
                        }
                    }
                    foreach (ActValues a in MovIngresos.Values)
                    {
                        if (activity.ActType == "Inversión" && a.Date.Month == mes)
                        {
                            a.Quantity += activity.Quantity;
                        }
                    }
                }
            }
            nombreIngresos.DisplayName = MovIngresos.Tipo;
            ingresosDatos.DataSource = MovIngresos.Values;
            nombreGastos.DisplayName = MovGastos.Tipo;
            gastosDatos.DataSource = MovGastos.Values;
        }
    }
}

```

Este método es muy parecido al que es para las gráficas de barras, pero aquí también se asigna la propiedad DisplayName para LineSeries, así cuando el usuario presione sobre un punto se le mostrará el tipo de movimiento junto a la cantidad del mes.

```
// Para la gráfica de comparación
5 referencias
public class Act
{
    3 referencias
    public string Tipo { get; }
    5 referencias
    public IList<ActValues> Values { get; }

    2 referencias
    public Act(string mov, params ActValues[] values)
    {
        this.Tipo = mov;
        this.Values = new List<ActValues>(values);
    }
}

30 referencias
public class ActValues
{
    3 referencias
    public DateTime Date { get; }
    3 referencias
    public double Quantity { get; set; }

    24 referencias
    public ActValues(DateTime month, double value)
    {
        this.Date = month;
        this.Quantity = value;
    }
}
```

La clase Act contiene las propiedades de Tipo (string) y Values (IList<ActValues>) y son inicializados en el constructor. La clase ActValues contiene la fecha y la cantidad total del movimiento que se ha realizado en ese mes.

Todos los métodos creados para cargar las gráficas son llamados en el método override OnAppearing y también se añade al evento de SelectedIndexChanged del Picker el método mesPie_SelectedIndexChanged, para cargar en la gráfica de secciones los gastos e inversiones del mes seleccionado por el usuario cuando este decida visualizar los movimientos de otro mes.

```
0 referencias
protected override void OnAppearing()
{
    LoadActivities();
    LoadMovimientosSeries();
    LoadComparacion();
    mesPie.SelectedIndexChanged += mesPie_SelectedIndexChanged;
}

1 referencia
private void mesPie_SelectedIndexChanged(object sender, EventArgs e)
{
    LoadActivities();
}
```

10. ActivitiesScreen

La pantalla de movimientos permitirá al usuario añadir un movimiento y debajo saldrán todos los que ha realizado. Cuando el usuario introduzca algún movimiento, se añade al stackLayout un AbsoluteLayout con un botón que muestra la información del movimiento. Y una adición respecto al diseño mostrado es un BoxView donde se mostrará el capital actual que tiene el usuario, que irá cambiando dependiendo de los movimientos que se realicen y sus modificaciones.

```
<StackLayout Margin="30, 30, 30, 0" x:Name="layout">
    <AbsoluteLayout Margin="0, 10, 0, 60">
        <BoxView AbsoluteLayout.LayoutBounds="0.5, 0, 330, 120"
            AbsoluteLayout.LayoutFlags="PositionProportional"
            Color="#90D490" CornerRadius="10">
            <BoxView.Shadow>
                <Shadow Brush="Gray"
                    Offset="0,15"
                    Radius="10"
                    Opacity="0.5"/>
            </BoxView.Shadow>
        </BoxView>
        <Label Text="CAPITAL ACTUAL" FontFamily="Montserrat-Medium"
            AbsoluteLayout.LayoutBounds="0.5, 0.1, -1, -1" FontSize="18"
            AbsoluteLayout.LayoutFlags="PositionProportional" />
        <Line X1="0" Y1="0" X2="330" Y2="0"
            StrokeLineCap="Flat" StrokeThickness="2.5" Stroke="black"
            AbsoluteLayout.LayoutBounds="-1, 0.35, -1, -1"
            AbsoluteLayout.LayoutFlags="PositionProportional"/>
        <Label x:Name="capActual" FontFamily="Montserrat-Medium"
            AbsoluteLayout.LayoutBounds="0.5, 0.8, -1, -1" FontSize="26"
            AbsoluteLayout.LayoutFlags="PositionProportional" />
    </AbsoluteLayout>
</StackLayout>
```

```
<AbsoluteLayout>
    <BoxView AbsoluteLayout.LayoutBounds="0.5, 0, 330, 58"
        AbsoluteLayout.LayoutFlags="PositionProportional"
        Color="#BCBCBC" CornerRadius="5, 5, -1, -1">
        <Label Text="Inversión" FontFamily="Montserrat-Medium"
            AbsoluteLayout.LayoutBounds="0.07, 0.2, -1, -1"
            AbsoluteLayout.LayoutFlags="PositionProportional" />
        <Entry x:Name="invest" FontSize="17" FontFamily="Montserrat-Medium"
            Keyboard="Numeric" MaxLength="10"
            AbsoluteLayout.LayoutBounds="0.5, 1.5, 305, 45"
            AbsoluteLayout.LayoutFlags="PositionProportional" />
    </AbsoluteLayout>
    <Button Text="+" Añadir" FontFamily="Montserrat-Medium" BackgroundColor="#BCBCBC" Margin="0, 20, 0, 40"
        TextColor="Black" CornerRadius="10" WidthRequest="100" HorizontalOptions="End" Clicked="AddInvestAsync"/>
    <AbsoluteLayout>
        <BoxView AbsoluteLayout.LayoutBounds="0.5, 0, 330, 58"
            AbsoluteLayout.LayoutFlags="PositionProportional"
            Color="#BCBCBC" CornerRadius="5, 5, -1, -1">
            <Label Text="Gasto" FontFamily="Montserrat-Medium"
                AbsoluteLayout.LayoutBounds="0.07, 0.2, -1, -1"
                AbsoluteLayout.LayoutFlags="PositionProportional" />
            <Entry x:Name="outlay" FontSize="17" FontFamily="Montserrat-Medium"
                Keyboard="Numeric" MaxLength="10"
                AbsoluteLayout.LayoutBounds="0.5, 1.5, 305, 45"
                AbsoluteLayout.LayoutFlags="PositionProportional" />
        </AbsoluteLayout>
        <Button Text="+" Añadir" FontFamily="Montserrat-Medium" BackgroundColor="#BCBCBC" Margin="0, 20, 0, 40"
            TextColor="Black" CornerRadius="10" WidthRequest="100" HorizontalOptions="End" Clicked="AddOutlayAsync"/>
        <Label Text="GASTOS E INVERSIONES" FontAttributes="Bold"
            FontFamily="Montserrat-Medium" FontSize="15"/>
        <Line X1="0" Y1="0" X2="330" Y2="0"
            HorizontalOptions="CenterAndExpand"
            StrokeLineCap="Flat" StrokeThickness="2.5" Stroke="black"/>
        <Button Text="Eliminar" Clicked="Button_Clicked"/>
    </AbsoluteLayout>
</StackLayout>
</ScrollView>
</ContentPage>
```

En el constructor de la clase se inicializa la base de datos, se oculta la barra de navegación, ya que ya está establecido en AppShell. Se crea una variable estática, donde se guardará el id del botón del movimiento que el usuario ha apretado y otra variable de tipo User para obtener el usuario de la base de datos. También bloqueamos esta pantalla para que el usuario no pueda utilizar el botón de retroceso del dispositivo móvil, esto se va a realizar en los tres apartados de la app.

```
private readonly CashFlowDatabase database;
public static string buttonId;
private User user;
0 references
public ActivitiesScreen()
{
    InitializeComponent();
    NavigationPage.SetHasNavigationBar(this, false);
    database = new CashFlowDatabase();
}

0 references
protected override bool OnBackButtonPressed()
{
    return true;
}
```

El método Load() se encarga de añadir los movimientos como elementos al StackLayout, obtiene de la base de datos todos los movimientos, y si no está vacío la lista se recorre en un foreach y se añade a StackLayout con el método AddElement(). También asigna el capital actual del usuario que hay en la etiqueta capActual para reflejar los cambios en el capital dependiendo de los movimientos.

```
2 references
private async void Load()
{
    List<Activities> activities = await database.GetActivitiesAsync();
    if (activities.Count > 0)
    {
        foreach (Activities activity in activities)
        {
            AddElement(activity);
        }
    }
    user = await database.GetUserAsync();
    capActual.Text = user.Capital.ToString() + " €";
}
```

Este método es llamado en el override de OnAppearing para que se produzca cada vez que se accede a la pantalla, dentro de este método también se borra los elementos a partir del índice 6 del StackLayout, ya que Load recargará todo de nuevo y si ha habido modificaciones, se verán reflejados.

```
0 references
protected override void OnAppearing()
{
    var content = (StackLayout)FindByName("layout");
    for (int i = content.Children.Count - 1; i >= 7; i--)
    {
        content.Children.RemoveAt(i);
    }
    Load();
}
```

Para los eventos de los botones, se definen los métodos para el botón de añadir una inversión (AddInvestAsync) y el de añadir un gasto (AddOutlayAsync). Los dos métodos son prácticamente iguales, se verifica que lo que se ha introducido en el entry para la inversión o el gasto es correcto, crea el movimiento, cada uno con su ActType y lo añade a la base de datos. Luego de añadir el movimiento modifica el capital que posee el usuario con el método privado UpdateCapital(). Muestra en una ventana al usuario de que se ha realizado la acción con éxito y se vacía los dos entries. Al añadir un gasto o inversión también hay que añadirlo al stackLayout para que el usuario lo pueda visualizar, para ello se emplea también el método privado AddElement().

```
0 references
private async void AddInvestAsync(object sender, EventArgs e)
{
    if (float.TryParse(invest.Text, out float result) && !invest.Text.StartsWith("-") && !string.IsNullOrWhiteSpace(result))
    {
        double inv = Convert.ToDouble(invest.Text, CultureInfo.InvariantCulture);
        Activities activity = new Activities
        {
            ActType = "Inversión",
            Quantity = (float)Math.Round(inv, 2),
            ActivityDate = DateTime.Now
        };
        await database.AddActivityAsync(activity);
        UpdateCapital(activity.Quantity, activity);
        AddElement(activity);
        await DisplayAlert("Exito", "Nueva inversión añadida correctamente", "Aceptar");
        invest.Text = "";
        outlay.Text = "";
        user = await database.GetUserAsync();
        capActual.Text = user.Capital.ToString() + " €";
    }
    else
    {
        await DisplayAlert("Error", "Error, introduzca la inversión de forma correcta", "Aceptar");
    }
}
```

```

private async void AddOutlayAsync(object sender, EventArgs e)
{
    if (float.TryParse(outlay.Text, out float result) && !outlay.Text.StartsWith("-") && !string.IsNullOrWhiteSpace(outlay.Text))
    {
        double outl = Convert.ToDouble(outlay.Text, CultureInfo.InvariantCulture);
        user = await database.GetUserAsync();
        if (outl < user.Capital)
        {
            Activities activity = new Activities
            {
                ActType = "Gasto",
                Quantity = (float)Math.Round(outl, 2),
                ActivityDate = DateTime.Now
            };
            await database.AddActivityAsync(activity);
            UpdateCapital(activity.Quantity, activity);
            AddElement(activity);
            await DisplayAlert("Exito", "Nuevo gasto añadida correctamente", "Aceptar");
            invest.Text = "";
            outlay.Text = "";
            user = await database.GetUserAsync();
            capActual.Text = user.Capital.ToString() + " €";
        }
        else
        {
            await DisplayAlert("Error", "Error, no le queda suficiente capital", "Aceptar");
        }
    }
    else
    {
        await DisplayAlert("Error", "Error, introduzca el gasto de forma correcta", "Aceptar");
    }
}

```

En este último método también se comprueba que el gasto es menor o igual que el capital, ya que en caso contrario no se podrá añadir el movimiento.

El método UpdateCapital pide como parámetros el capital y el movimiento de Activities, verifica que tipo es el movimiento y ahí resta o suma el capital del usuario y lo actualiza en la base de datos.

```

private async void UpdateCapital(float cap, Activities act)
{
    user = await database.GetUserAsync();
    if(act.ActType == "Inversión")
    {
        User editedUser = new User
        {
            Id = user.Id,
            Name = user.Name,
            Surnames = user.Surnames,
            InitCapital = user.InitCapital,
            Capital = user.Capital + cap,
            MensualEarning = user.MensualEarning,
            NamePrivkey = user.NamePrivkey,
            SurnamesPrivKey = user.SurnamesPrivKey
        };
        await database.UpdateUserAsync(editedUser);
    }
    else
    {
        User editedUser = new User
        {
            Id = user.Id,
            Name = user.Name,
            Surnames = user.Surnames,
            InitCapital = user.InitCapital,
            Capital = user.Capital - cap,
            MensualEarning = user.MensualEarning,
            NamePrivkey = user.NamePrivkey,
            SurnamesPrivKey = user.SurnamesPrivKey
        };
        await database.UpdateUserAsync(editedUser);
    }
}

```

El método privado AddElement pide como parámetro un movimiento de tipo Activities, verifica si se trata de una inversión o de un gasto y una vez verificado se crea un AbsoluteLayout donde se añadirá un botón y una label. Se crea el botón con sus propiedades, asignándole al evento Clicked el método Acciones, después de haber asignado todas las propiedades se le añade al AbsoluteLayout. Por último se crea una etiqueta que muestra el tipo de movimiento que es (inversión o gasto) y se añade al AbsoluteLayout. Una vez añadido los elementos al AbsoluteLayout creado previamente, este es añadido al StackLayout con nombre “layout”, donde se encuentran los demás elementos de la pantalla.

```
3 referencias
private void AddElement(Activities activity)
{
    string texto = "";
    if (activity.ActType == "Inversión")
    {
        texto = "+" + activity.Quantity + "€";
        texto = texto.PadRight(50) + activity.ActivityDate.ToShortDateString();
    }
    else
    {
        texto = "- " + activity.Quantity + "€";
        texto = texto.PadRight(50) + activity.ActivityDate.ToShortDateString();
    }
    AbsoluteLayout element = new AbsoluteLayout { Margin = new Thickness(0, 20, 0, 5)};
    var content = (StackLayout)FindByName("layout");
    Button button = new Button
    {
        AutomationId = activity.Id.ToString(),
        FontFamily = "Montserrat-Medium",
        BackgroundColor = Color.FromArgb("#DEDEDE"),
        TextColor = Color.FromRgb(0, 0, 0),
        Text = texto,
        CornerRadius = 10,
        Padding = new Thickness(0, 20, 0, 0),
    };
    button.Clicked += Acciones;
    button.Shadow = new Shadow()
    {
        Brush = new SolidColorBrush(Color.FromRgb(0, 0, 0)),
        Offset = new Point(10, 10),
        Radius = 10,
        Opacity = (Float)0.3
    };
    AbsoluteLayout.SetLayoutBounds(button, new Rect(0.5, 0, 320, 67));
    AbsoluteLayout.SetLayoutFlags(button, Microsoft.Maui.Layouts.AbsoluteLayoutFlags.PositionProportional);
    element.Children.Add(button);
    Label label = new Label { Text = activity.ActType, FontAttributes = FontAttributes.Bold, FontFamily = "M
    AbsoluteLayout.SetLayoutBounds(label, new Rect(8, 0.4, 330, 40));
    AbsoluteLayout.SetLayoutFlags(label, Microsoft.Maui.Layouts.AbsoluteLayoutFlags.PositionProportional);
    element.Children.Add(label);
    content.Children.Add(element);
}
```

Como se ve, el elemento creado es un AbsoluteLayout que contiene un botón y una label, por lo que si el usuario presiona sobre el elemento la acción Clicked se ejecutará y por eso se le ha asignado el método Acciones(). Este método muestra en una ventana, con el método DisplayActionSheet, al usuario la opción de editar o eliminar el movimiento y la opción que elija se guarda en una variable y dependiendo de la opción elegida se realiza una acción u otra. Si elige editar primero se le asigna a la variable estática buttonId el AutomationId del botón presionado y manda al usuario a la página para la edición del movimiento. Por otro lado si el usuario decide eliminar el movimiento, primero se obtiene el movimiento de la tabla Activities y se elimina de la base de datos y para que el elemento sea eliminado primero se elimina todos los elementos que muestran las inversiones y gastos y se llama al método Load() para que vuelva a cargar los movimientos, esta vez sin el que ha sido eliminado. También se debe modificar el capital del usuario, así que dependiendo de qué tipo de inversión se elimina, sumará o restará a el capital. Una vez terminada la acción se le mostrará al usuario que se ha eliminado correctamente y también se vacían los entries por si contienen algo.

```
1 reference
private async void Acciones(object sender, EventArgs e)
{
    string opcion = await DisplayActionSheet("Elija una opción", "Salir", null, "Editar", "Eliminar"
    Button btn = (Button)sender;

    if (opcion == "Editar")
    {
        buttonId = btn.AutomationId;
        await Navigation.PushAsync(new ActivityEditScreen());
    }
    else if (opcion == "Eliminar")
    {
        Activities activity = await database.GetActivityAsync(int.Parse(btn.AutomationId));
        await database.DeleteActivityAsync(activity);
        var content = (StackLayout)FindByName("layout");
        for (int i = content.Children.Count - 1; i >= 7; i--)
        {
            content.Children.RemoveAt(i);
        }
        if (activity.ActType == "Inversión")
        {
            activity.ActType = "Gasto";
            UpdateCapital(activity.Quantity, activity);
        }
        else
        {
            activity.ActType = "Inversión";
            UpdateCapital(activity.Quantity, activity);
        }
        await DisplayAlert("Éxito", "Movimiento eliminado correctamente", "Aceptar");
        invest.Text = "";
        outlay.Text = "";
        Load();
    }
}
```

11. ActivityEditScreen

Esta página es al que accede el usuario cuando quiere editar un movimiento. Hay un Picker, donde el usuario puede cambiar el tipo de movimiento (inversión o gasto), un Entry donde puede cambiar la cantidad de dinero afectado en el movimiento y un DatePicker para cambiar la fecha que estará limitada al día en el que se encuentra el usuario en el momento de la edición. Y debajo hay dos botones, uno para cancelar la acción, que devolverá al usuario a la página anterior y otro botón para guardar los cambios que ha introducido el usuario. Esta pantalla sí que permite al usuario utilizar el botón de retroceso del teléfono móvil para volver al igual que cuando se edita un movimiento.

```
<ScrollView>
    <StackLayout Margin="30, 55, 30, 0">
        <Label Text="Tipo de movimiento" FontSize="15"/>
        <Picker Title="Tipo de movimiento" x:Name="tipoMovimiento" Margin="0, 0, 0, 40" FontSize="18">
            <Picker.ItemsSource>
                <x:Array Type="{x:Type x:String}">
                    <x:String>Inversión</x:String>
                    <x:String>Gasto</x:String>
                </x:Array>
            </Picker.ItemsSource>
        </Picker>
        <Label Text="Cantidad()" FontFamily="Montserrat-Medium" FontSize="15"/>
        <AbsoluteLayout Margin="0, 10, 0, 40">
            <BoxView AbsoluteLayout.LayoutBounds="0.5, 0, 330, 40"
                    AbsoluteLayout.LayoutFlags="PositionProportional"
                    Color="LightGray" CornerRadius="5, 5, -1, -1" Opacity="0.8"/>
            <Entry x:Name="invest" FontSize="17" FontFamily="Montserrat-Medium"
                   Keyboard="Numeric" MaxLength="12"
                   AbsoluteLayout.LayoutBounds="0.5, 1.5, 320, 45"
                   AbsoluteLayout.LayoutFlags="PositionProportional" />
        </AbsoluteLayout>
    </StackLayout>
    <Label Text="Fecha del movimiento" FontFamily="Montserrat-Medium" FontSize="15"/>
    <DatePicker x:Name="fechaMov" Margin="0, 0, 0, 20" FontSize="18"/>
    <HorizontalStackLayout Margin="0, 50, 0, 10" Spacing="120" HorizontalOptions="Center">
        <Button Text="Cancelar" FontFamily="Montserrat-Medium" BackgroundColor="LightGray"
               TextColor="Black" CornerRadius="10" WidthRequest="100" Opacity="0.8" Clicked="CancelarAsync"/>
        <Button Text="Aceptar" FontFamily="Montserrat-Medium" BackgroundColor="LightGray"
               TextColor="Black" CornerRadius="10" WidthRequest="100" Opacity="0.8" Clicked="GuardarCambios"/>
    </HorizontalStackLayout>
</StackLayout>
</ScrollView>
```

En cuanto al código de esta pantalla, se inicializa la base de datos para poder actualizar el elemento de la tabla Activities. Pero primero de todo se llama al método CargarDatos para llenar los datos que tiene el movimiento que se quiere editar en los Entries de la pantalla de edición, para mostrarlos al usuario y este pueda tenerlo como guía para los cambios que vaya a realizar.

```
private readonly CashFlowDatabase database;
1 reference
public ActivityEditScreen()
{
    InitializeComponent();
    NavigationPage.SetHasNavigationBar(this, false);
    database = new CashFlowDatabase();
    CargarDatos();
}

1 reference
private async void CargarDatos()
{
    Activities activity = await database.GetActivityAsync(int.Parse(ActivityScreen.buttonId));
    if(activity.ActType == "Inversión")
    {
        tipoMovimiento.SelectedIndex = 0;
    }
    else
    {
        tipoMovimiento.SelectedIndex = 1;
    }
    invest.Text = activity.Quantity.ToString();
    fechaMov.Date = activity.ActivityDate;
}
```

Una vez hecho el método CargarDatos, habrá que completar los dos métodos para el evento del botón de Cancelar y Guardar. Para el botón Cancelar simplemente se mostrará al usuario si de verdad quiere cancelar la acción y si así es volverá a la pantalla de los movimientos.

```
0 references
private async void CancelAsync(object sender, EventArgs e)
{
    bool respuesta = await DisplayAlert("Cancelar", "¿Desea continuar?", "Sí", "No");
    if(respuesta)
    {
        await Navigation.PopAsync();
    }
}
```

Y para el botón guardar, el método GuardarCambios cogerá toda los datos del formulario (verificándolos tambien), creará un nuevo movimiento de tipo Activities con el mismo id que el que se quiere editar, este id se obtiene llamando a la variable estática de la clase ActivitiesScreen “buttonId”. Más tarde, se usa el método UpdateActivity de la base de datos introduciendo el movimiento como parámetro, luego se muestra al usuario que ha sido actualizado correctamente y le manda de vuelta a la página de Movimientos. Por ultimo será necesario actualizar el usuario la capital y dependiendo de las ediciones que haga el usuario.

```
private async void GuardarCambios(object sender, EventArgs e)
{
    if(float.TryParse(invest.Text, out float result) && !invest.Text.StartsWith("-") && !string.IsNullOrEmpty(result.ToString()))
    {
        double inv = Convert.ToDouble(invest.Text, CultureInfo.InvariantCulture);
        User user = await database.GetUserAsync();
        Activities oldActivity = await database.GetActivityAsync(int.Parse(ActivitiesScreen.buttonId));
        Activities activity = new Activities
        {
            Id = int.Parse(ActivitiesScreen.buttonId),
            ActType = tipoMovimiento.SelectedItem.ToString(),
            Quantity = (float)Math.Round(inv, 2),
            ActivityDate = fechaMov.Date
        };
        if (activity.ActType == "Gasto" && oldActivity.ActType == "Inversión")
        {
            User editedUser = new User
            {
                Id = user.Id,
                Name = user.Name,
                Surnames = user.Surnames,
                InitCapital = user.InitCapital,
                Capital = user.Capital - activity.Quantity - oldActivity.Quantity,
                MensualEarning = user.MensualEarning,
                NamePrivateKey = user.NamePrivateKey,
                SurnamesPrivateKey = user.SurnamesPrivateKey
            };
            await database.UpdateUserAsync(editedUser);
        }
        else if (activity.ActType == "Inversión" && oldActivity.ActType == "Gasto")
        {
            User editedUser = new User
            {
                Id = user.Id,
                Name = user.Name,
                Surnames = user.Surnames,
                InitCapital = user.InitCapital,
                Capital = user.Capital + activity.Quantity + oldActivity.Quantity,
                MensualEarning = user.MensualEarning,
                NamePrivateKey = user.NamePrivateKey,
                SurnamesPrivateKey = user.SurnamesPrivateKey
            };
            await database.UpdateUserAsync(editedUser);
        }
    }
}
```

```
        else
        {
            if(activity.ActType == "Inversión")
            {
                User editedUser = new User
                {
                    Id = user.Id,
                    Name = user.Name,
                    Surnames = user.Surnames,
                    InitCapital = user.InitCapital,
                    Capital = user.Capital - oldActivity.Quantity + activity.Quantity,
                    MensualEarning = user.MensualEarning,
                    NamePrivkey = user.NamePrivkey,
                    SurnamesPrivKey = user.SurnamesPrivKey
                };
                await database.UpdateUserAsync(editedUser);
            }
            else
            {
                User editedUser = new User
                {
                    Id = user.Id,
                    Name = user.Name,
                    Surnames = user.Surnames,
                    InitCapital = user.InitCapital,
                    Capital = user.Capital + oldActivity.Quantity - activity.Quantity,
                    MensualEarning = user.MensualEarning,
                    NamePrivkey = user.NamePrivkey,
                    SurnamesPrivKey = user.SurnamesPrivKey
                };
                await database.UpdateUserAsync(editedUser);
            }
        }
        await database.UpdateActivity(activity);
        await DisplayAlert("Éxito", "Modificado el movimiento correctamente", "Aceptar");
        await Navigation.PopAsync();
    }
    else
    {
        await DisplayAlert("Error", "Error, al actualizar, compruebe los datos", "Aceptar");
    }
}
```

12. ConfigurationScreen

Por último, la pantalla de configuración. Muestra toda la información del usuario y debajo varios botones, un botón para editar el perfil, otro para eliminar todos los movimientos realizados y uno último para eliminar el usuario.

```
<ScrollView>
    <StackLayout Margin="30, 25, 30, 30">
        <Label Text="Información del usuario" FontFamily="Montserrat-Medium"
               FontSize="21" HorizontalOptions="Center" Margin="0, 10, 0, 35"/>
        <Line X1="0" Y1="0" X2="360" Y2="0"
              StrokeThickness="2" Stroke="Gray"/>
        <StackLayout Margin="10" Orientation="Horizontal">
            <Label Text="Nombre:" FontFamily="Montserrat-Medium"
                   FontSize="18" HorizontalOptions="StartAndExpand"/>
            <Label x:Name="nombre" FontFamily="Montserrat-Medium"
                   FontSize="18"/>
        </StackLayout>

        <Line X1="0" Y1="0" X2="360" Y2="0" HeightRequest="30"
              StrokeLineCap="Flat" StrokeThickness="2" Stroke="Gray"/>

        <Line X1="0" Y1="0" X2="360" Y2="0"
              StrokeLineCap="Flat" StrokeThickness="2" Stroke="Gray"/>

        <StackLayout Margin="10" Orientation="Horizontal">
            <Label Text="Apellidos:" FontFamily="Montserrat-Medium"
                   FontSize="18" HorizontalOptions="StartAndExpand"/>
            <Label x:Name="apellidos" FontFamily="Montserrat-Medium"
                   FontSize="18"/>
        </StackLayout>

        <Line X1="0" Y1="0" X2="360" Y2="0" HeightRequest="30"
              StrokeLineCap="Flat" StrokeThickness="2" Stroke="Gray"/>

        <Line X1="0" Y1="0" X2="360" Y2="0"
              StrokeLineCap="Flat" StrokeThickness="2" Stroke="Gray"/>

        <StackLayout Margin="10" Orientation="Horizontal">
            <Label Text="Capital:" FontFamily="Montserrat-Medium"
                   FontSize="18" HorizontalOptions="StartAndExpand"/>
            <Label x:Name="capital" FontFamily="Montserrat-Medium"
                   FontSize="18"/>
        </StackLayout>

        <Line X1="0" Y1="0" X2="360" Y2="0" HeightRequest="30"
              StrokeLineCap="Flat" StrokeThickness="2" Stroke="Gray"/>

        <Line X1="0" Y1="0" X2="360" Y2="0"
              StrokeLineCap="Flat" StrokeThickness="2" Stroke="Gray"/>

        <StackLayout Margin="10" Orientation="Horizontal">
            <Label Text="Ganancia mensual:" FontFamily="Montserrat-Medium"
                   FontSize="18" HorizontalOptions="StartAndExpand"/>
            <Label x:Name="gananciaM" FontFamily="Montserrat-Medium"
                   FontSize="18"/>
        </StackLayout>
    </StackLayout>
```

```
<Line X1="0" Y1="0" X2="360" Y2="0" HeightRequest="30"
      StrokeLineCap="Flat" StrokeThickness="2" Stroke="Gray"/>
<Button Text="Editar perfil" FontFamily="Montserrat-Medium" BackgroundColor="LightGray" Margin="0, 20, 0, 10"
        TextColor="Black" CornerRadius="10" WidthRequest="200" Clicked="EditProfile"/>
<Button Text="Eliminar movimientos" FontFamily="Montserrat-Medium" BackgroundColor="LightGray" Margin="0, 20, 0, 10"
        CornerRadius="10" WidthRequest="200" TextColor="Red" Clicked="DeleteActivities"/>
<Button Text="Eliminar usuario" FontFamily="Montserrat-Medium" BackgroundColor="LightGray" Margin="0, 20, 0, 10"
        CornerRadius="10" WidthRequest="200" TextColor="Red" Clicked="DeleteUser"/>
</StackLayout>
```

En cuanto al código C#, en el constructor se oculta el encabezado y se inicializa la base de datos, y también se bloquea el botón de retroceso.

```

    private readonly CashFlowDatabase database;
    O references
    public ConfigScreen()
    {
        InitializeComponent();
        NavigationPage.SetHasNavigationBar(this, false);
        database = new CashFlowDatabase();
    }
    O references
    protected override bool OnBackButtonPressed()
    {
        return true;
    }

```

Se usará el método override OnAppearing para llamar al método LoadData() que se encargará de cargar todos los datos del usuario y mostrarlas en la pantalla en las Labels.

```

    1 reference
    private async void LoadData()
    {
        User user = await database.GetUserAsync();
        nombre.Text = RSAUtils.Desencriptar(user.NamePrivkey, user.Name);
        apellidos.Text = RSAUtils.Desencriptar(user.SurnamesPrivKey, user.Surnames);
        capital.Text = user.Capital.ToString(CultureInfo.InvariantCulture);
        gananciaM.Text = user.MensualEarning.ToString(CultureInfo.InvariantCulture);
    }

    O references
    protected override void OnAppearing()
    {
        LoadData();
    }

```

En cuanto los botones, el de editar tendrá como evento EditProfile que le llevará al usuario a la pantalla de ConfigScreenEdit, para el botón de eliminar todos los movimientos el método DeleteActivities() y para eliminar el usuario DeleteUser().

```

    O references
    private async void EditProfile(object sender, EventArgs e)
    {
        await Navigation.PushAsync(new ConfigScreenEdit());
    }

    O references
    private async void DeleteActivities(object sender, EventArgs e)
    {
        bool respuesta = await DisplayAlert("Eliminar movimientos", "Está a punto de eliminar todos los movimientos de la cuenta. ¿Quiere continuar?", "Cancelar", "Aceptar");
        if (respuesta)
        {
            await database.DeleteAllActivities();
            User oldUser = await database.GetUserAsync();
            User user = new User()
            {
                Id = oldUser.Id,
                Name = oldUser.Name,
                Surnames = oldUser.Surnames,
                InitCapital = oldUser.InitCapital,
                Capital = oldUser.InitCapital,
                NamePrivkey = oldUser.NamePrivkey,
                SurnamesPrivKey = oldUser.SurnamesPrivKey
            };
            await database.UpdateUserAsync(user);
            capital.Text = user.Capital.ToString(CultureInfo.InvariantCulture);
            await DisplayAlert("Eliminado", "Se han eliminado todos los movimientos que existian", "Aceptar");
        }
    }

```

Para eliminar todos los movimientos de la base de datos se usa el método DeleteAllActivities, donde también se deberá actualizar al usuario para que vuelva a tener como capital el capital inicial.

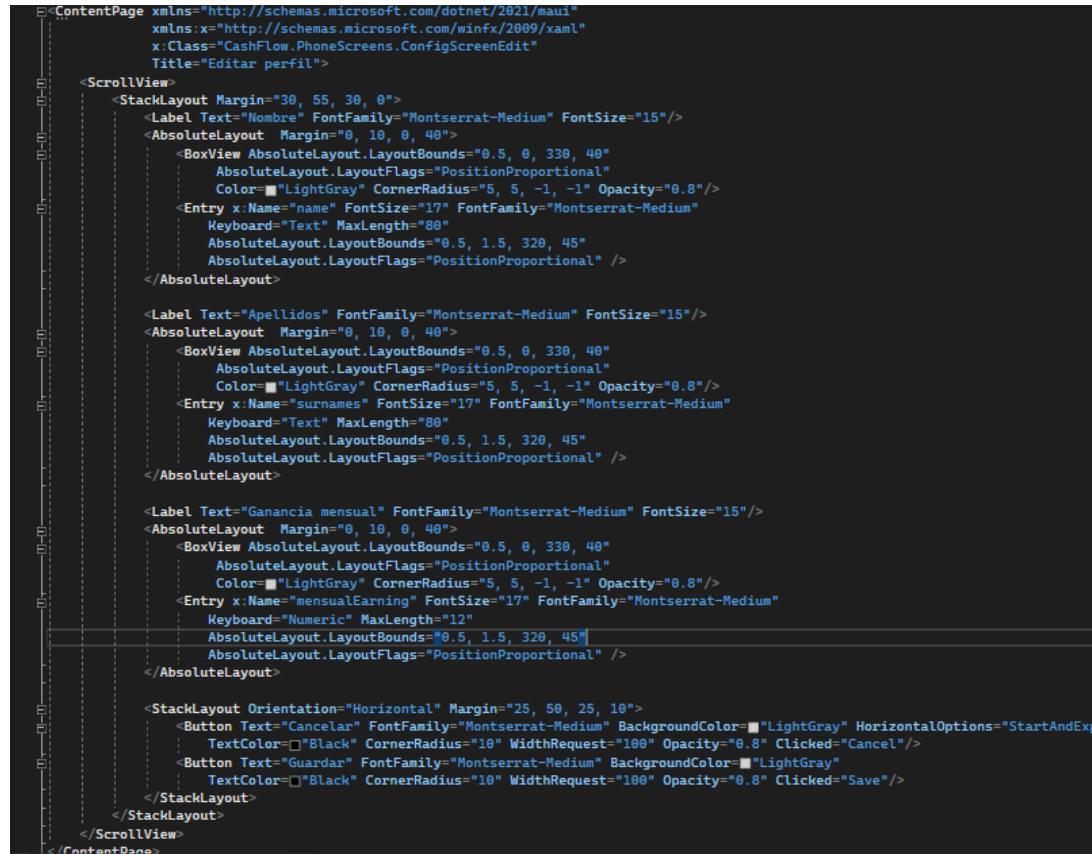
```
0 references
private async void DeleteActivities(object sender, EventArgs e)
{
    bool respuesta = await DisplayAlert("Eliminar movimientos", "Está a punto de eliminar todos los movimientos", "Aceptar", "Cancelar");
    if (respuesta)
    {
        await database.DeleteAllActivities();
        User oldUser = await database.GetUserAsync();
        User user = new User()
        {
            Id = oldUser.Id,
            Name = oldUser.Name,
            Surnames = oldUser.Surnames,
            InitCapital = oldUser.InitCapital,
            Capital = oldUser.InitCapital,
            NamePrivkey = oldUser.NamePrivkey,
            SurnamesPrivKey = oldUser.SurnamesPrivKey
        };
        await database.UpdateUserAsync(user);
        capital.Text = user.Capital.ToString(CultureInfo.InvariantCulture);
        await DisplayAlert("Eliminado", "Se han eliminado todos los movimientos que existian", "Aceptar");
    }
}
```

```
0 references
private async void DeleteUser(object sender, EventArgs e)
{
    bool respuesta = await DisplayAlert("Eliminar usuario", "Al eliminar el usuario también se eliminarán todos sus movimientos", "Aceptar", "Cancelar");
    if (respuesta)
    {
        await database.DeleteUserAsync();
        await database.DeleteAllActivities();
        await DisplayAlert("Eliminado todo", "Se ha eliminado el usuario junto a todos sus movimientos", "Aceptar");
        Application.Current.Quit();
    }
}
```

En DeleteUser se elimina el usuario de la base de datos y esto supone también eliminar todos los movimientos de la tabla Activities y una vez terminado el proceso se cerrará la aplicación. Y si el usuario vuelve a entrar deberá registrarse de nuevo.

13. ConfigScreenEdit

Esta pantalla contiene los Entries con los datos ya cargados del usuario y le permite modificarlo. Y debajo se encuentran los botones de cancelar y guardar.



En el código de C# se inicializa en el constructor la base de datos y se llama al método LoadData para cargar los datos del usuario en los Entries.

```

private readonly CashFlowDatabase database;
1 reference
public ConfigScreenEdit()
{
    InitializeComponent();
    database = new CashFlowDatabase();
    LoadData();
}

1 reference
private async void LoadData()
{
    User user = await database.GetUserAsync();
    name.Text = RSAUtils.Desencriptar(user.NamePrivkey, user.Name);
    surnames.Text = RSAUtils.Desencriptar(user.SurnamesPrivKey, user.Surnames);
    mensualEarning.Text = user.MensualEarning.ToString(CultureInfo.InvariantCulture);
}

```

Para finalizar, los métodos Cancel y Save para los eventos de los botones de cancelar y guardar. Cancel() simplemente retrocede a la página anterior y Save() recoge de nuevo todos los datos de los entries, crea un nuevo User y con el método UpdateUserAsync de la base de datos se actualiza los datos del usuario.

```
private async void Cancel(object sender, EventArgs e)
{
    bool respuesta = await DisplayAlert("Titulo", "¿Desea continuar?", "Si", "No");
    if (respuesta)
    {
        await Navigation.PopAsync();
    }
}

0 referencias
private async void Save(object sender, EventArgs e)
{
    if(!string.IsNullOrWhiteSpace(name.Text) && !string.IsNullOrWhiteSpace(surnames.Text) && !mensualEarning.Text.IsNullOrWhiteSpace())
    {
        User oldUser = await database.GetUserAsync();
        string nombreEncriptado = RSAUtils.encriptar(name.Text.Trim());
        string namePrivKey = RSAUtils.privKeyStr;
        string apellidosEncriptado = RSAUtils.encriptar(surnames.Text.Trim());
        string surnamesPrivkey = RSAUtils.privKeyStr;

        double menE = Convert.ToDouble(mensualEarning.Text, CultureInfo.InvariantCulture);
        User user = new User()
        {
            Id = oldUser.Id,
            Name = nombreEncriptado,
            Surnames = apellidosEncriptado,
            InitCapital = oldUser.InitCapital,
            Capital = oldUser.Capital,
            MensualEarning = (float)Math.Round(menE, 2),
            NamePrivateKey = namePrivKey,
            SurnamesPrivateKey = surnamesPrivkey
        };
        await database.UpdateUserAsync(user);
        await DisplayAlert("Exito", "Modificado el movimiento correctamente", "Aceptar");
        await Navigation.PopAsync();
    }
    else
    {
        await DisplayAlert("Error", "Error, al actualizar, compruebe los datos", "Aceptar");
    }
}
```

Como se puede observar, se vuelve a recoger los datos de los entries, volviendo a encriptar el nombre y los apellidos al igual que se guardan las nuevas claves privadas de cada uno.

NOTA: Para finalizar, también se recomienda modificar el archivo del proyecto, eliminando todas las plataformas excepto Android e iOS, ya que **DevExpress** solo es compatible para esas dos plataformas. Con esta modificación no habrá problemas de compatibilidad y gracias a esto también se podrá borrar las carpetas de Windows, Tize y MacCatalyst de la carpeta Platforms. Si se desea no habría problema con eliminar también las dependencias de iOS y su plataforma ya que CashFlow está solo desarrollado y probado en Android, pero en este caso se va a conservar por si en un futuro se quiere implementar. El archivo CashFlow.csproj quedaría así en la parte de arriba:

```
<PropertyGroup>
    <TargetFrameworks>net7.0-android;net7.0-ios</TargetFrameworks>
    <!-- Uncomment to also build the tizen app. You will need to install tizen by following this: https://git
    <!-- <TargetFrameworks>$(TargetFrameworks);net7.0-tizen</TargetFrameworks> -->
    <OutputType>Exe</OutputType>
    <RootNamespace>CashFlow</RootNamespace>
    <UseMaui>true</UseMaui>
    <SingleProject>true</SingleProject>
    <ImplicitUsings>enable</ImplicitUsings>

    <!-- Display name -->
    <ApplicationTitle>CashFlow</ApplicationTitle>

    <!-- App Identifier -->
    <ApplicationId>com.companyname.cashflow</ApplicationId>
    <ApplicationIdGuid>8966ba13-330a-4e12-af5b-1039b3aed252</ApplicationIdGuid>

    <!-- Versions -->
    <ApplicationDisplayVersion>1.0</ApplicationDisplayVersion>
    <ApplicationVersion>1</ApplicationVersion>

    <SupportedOSPlatformVersion Condition="&lt;[MSBuild]::GetTargetPlatformIdentifier('$(TargetFramework)')&gt; == 
    <SupportedOSPlatformVersion Condition="&lt;[MSBuild]::GetTargetPlatformIdentifier('$(TargetFramework)')&gt; == 
</PropertyGroup>
```

Es simplemente eliminar las líneas relacionadas con las otras plataformas, no se requiere ninguna modificación más.

3.5. Pruebas

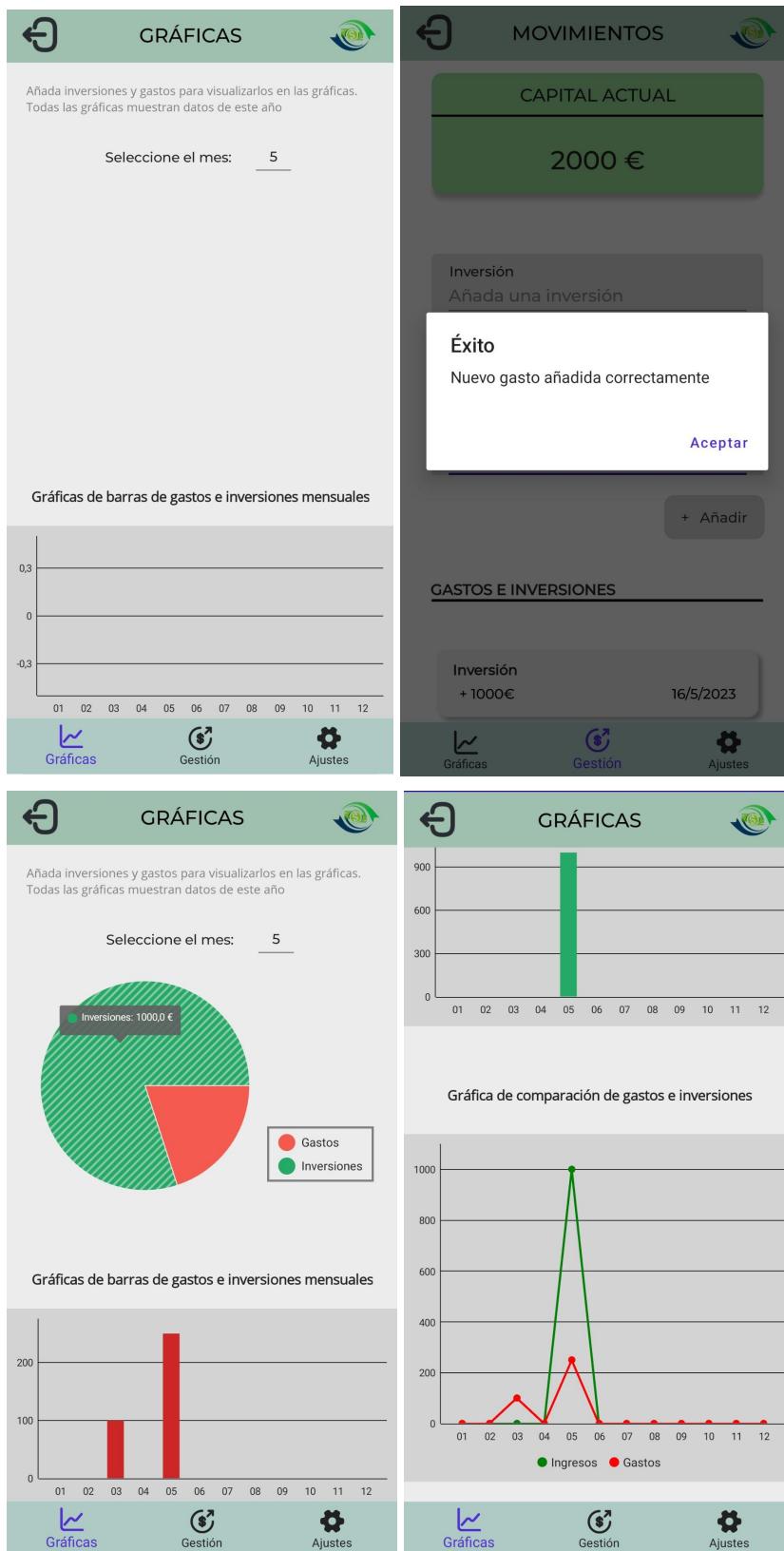
Por último, pero no menos importante, antes de dar por terminado el proyecto, se han realizado testeos de CashFlow para verificar que todo funciona de forma correcta, rápida y sin fallos muy grandes. Se han realizado pruebas para intentar detener la aplicación y comprobar así cómo se desenvuelve ante diferentes situaciones. Unos ejemplos han sido.

- Pantalla de registro: la pantalla verifica los datos correctamente y solo se activa el botón cuando los datos del Entry son correctos.

The image consists of four screenshots of a mobile application's registration screen, arranged in a 2x2 grid. Each screenshot shows a gradient background from light blue at the top to dark blue at the bottom. At the top center, the text "BIENVENIDO A CASHFLOW!" is displayed. Below it, a instruction "Regístrate para comenzar a gestionar su capital" is shown. There are four text input fields: "Jiacheng", "Lin", "Capital inicial*", and "Ganancia mensual". A large blue "REGISTRARSE" button is at the bottom.

- Screenshot 1:** All fields ("Jiacheng", "Lin", "Capital inicial*", "Ganancia mensual") contain valid input (non-empty strings).
- Screenshot 2:** The "Capital inicial*" field contains an invalid input ("-" instead of a number).
- Screenshot 3:** The "Ganancia mensual" field contains an invalid input ("1000" instead of a decimal like ".1000").
- Screenshot 4:** The "Ganancia mensual" field contains an invalid input ("." instead of a decimal like ".1000").

- Pantalla de gráficas: comprobar que no hay ningún error en las gráficas aunque no haya movimientos e introducir movimientos para verificar si se actualiza correctamente.

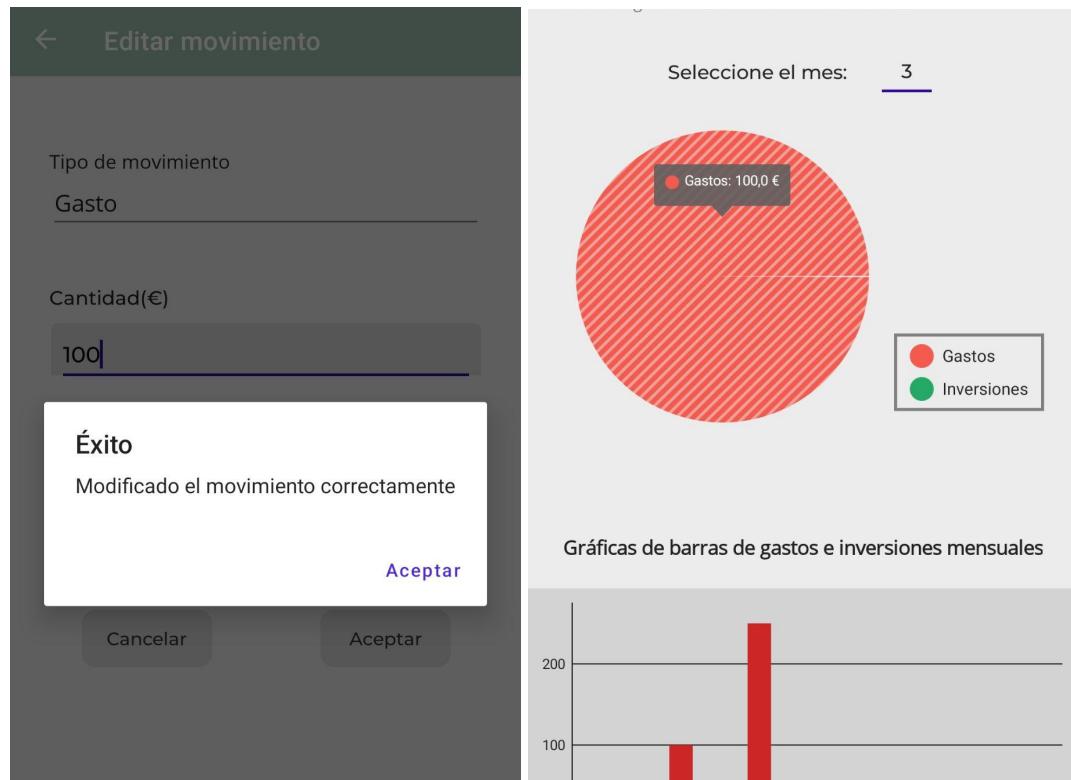


- Pantalla de movimientos: comprobar que se puede eliminar un movimiento y editar uno.

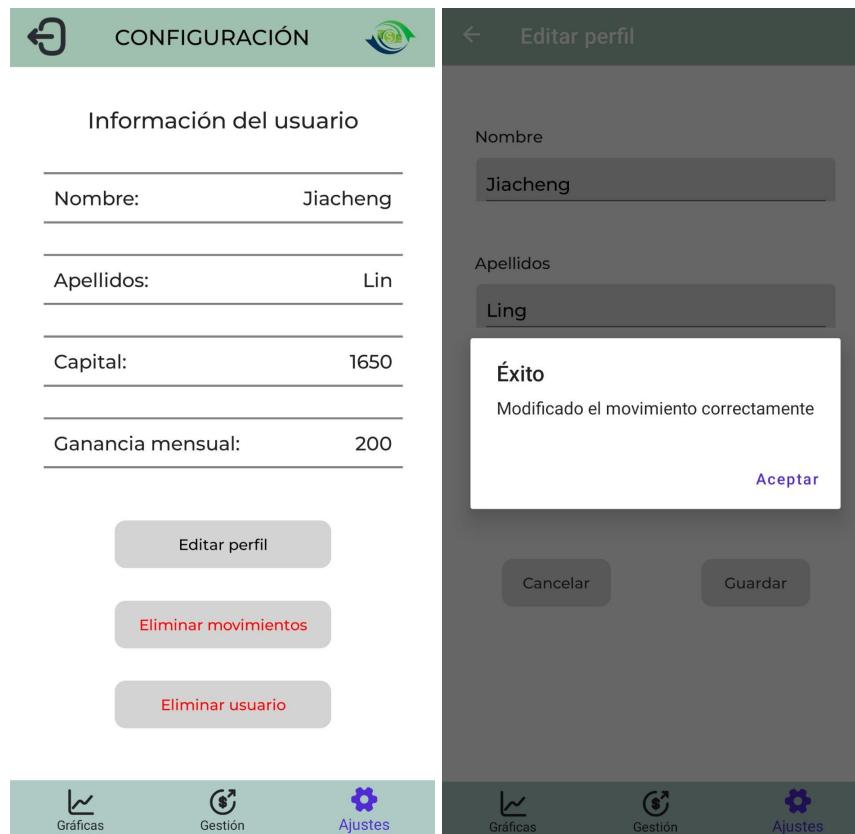
The image displays four screenshots of the CashFlow mobile application interface, illustrating various features related to movement management:

- Screenshot 1 (Top Left):** Shows a modal dialog titled "Elija una opción" (Select an option) with three options: "Editar" (Edit), "Eliminar" (Delete), and "Salir" (Exit). This is overlaid on a main screen showing a list of movements. At the top right of the main screen, the balance is displayed as 1870 €. Below the balance, there is a section for "Inversión".
- Screenshot 2 (Top Right):** Shows a confirmation message "Éxito" (Success) stating "Movimiento eliminado correctamente" (Movement deleted correctly). There is a "Aceptar" (Accept) button at the bottom right. The background shows the same main screen with the balance and investment section.
- Screenshot 3 (Bottom Left):** Shows the "Editar movimiento" (Edit movement) screen for an "Inversión" (Investment). The movement details are: Cantidad(€) 1200, Fecha del movimiento 16/5/2023. The "Cancelar" (Cancel) and "Aceptar" (Accept) buttons are at the bottom.
- Screenshot 4 (Bottom Right):** Shows the "Editar movimiento" (Edit movement) screen for a "Gasto" (Expense). The movement details are: Cantidad(€) 100, Fecha del movimiento 13/3/2023. The "Cancelar" (Cancel) and "Aceptar" (Accept) buttons are at the bottom.

At the bottom of each screenshot, there is a navigation bar with three icons: "Gráficas" (Graphs), "Gestión" (Management), and "Ajustes" (Settings).



- Pantalla de configuración: verificar el funcionamiento de los tres botones.



The image displays four screenshots of the CashFlow mobile application:

- Top Left (Configuration Screen):** Shows "Información del usuario" (User Information) with fields for "Nombre" (Name) and "Apellidos" (Last Name). Below this is a modal for "Eliminar movimientos" (Delete movements) asking if the user wants to delete all account movements. Buttons include "No", "Sí", "Editar perfil" (Edit profile), "Eliminar movimientos" (Delete movements), and "Eliminar usuario" (Delete user).
- Top Right (Movements Screen):** Shows "MOVIMIENTOS" (Movements) with a green header "CAPITAL ACTUAL" (Current Capital) displaying "1000 €". Below are sections for "Inversión" (Investment) and "Gasto" (Expense), each with a "+ Añadir" (Add) button.
- Bottom Left (Configuration Screen):** Similar to the top left, showing "Información del usuario" and the same modal for "Eliminar usuario".
- Bottom Right (Registration Screen):** Shows a large logo with a dollar sign and a circular arrow. The text "BIENVENIDO A CASHFLOW!" (Welcome to CashFlow!) is displayed. It includes fields for "Nombre*" (Name*), "Apellidos*" (Last Name*), "Capital inicial*" (Initial capital*), and "Ganancia mensual" (Monthly profit). A "REGISTRARSE" (Register) button is at the bottom, and a note at the bottom states: "Una vez registrado, la próxima vez que entre a la aplicación, accederá directamente al Inicio de CashFlow, recuerde que todo se guarda en el local" (Once registered, the next time you enter the application, you will directly access the CashFlow start screen, remember that everything is saved locally).

4. Conclusiones

Se ha conseguido desarrollar una aplicación con un ícono propio y una interfaz simple y sencilla para el uso de cualquier usuario de cualquier edad. Guarda los nombres y apellidos del usuario de forma encriptada a su vez que la clave privada de cada uno. Por otro lado también guarda su capital inicial, el capital actual y la ganancia mensual, este último es simplemente un dato que se muestra al usuario, pero se podría implementar a la aplicación de forma que cada día 1 del mes se realice una inversión de esa cantidad. Pero esto es solo una idea que no se ha conseguido por dificultad y por tiempo.

En general, la aplicación es funcional, muestra en gráficas los movimientos realizados que son útiles para el usuario, permite introducir movimientos y también muestra todos los que se han realizado, que pueden ser eliminados o editados por el usuario. Por último la pantalla de configuración que muestra la información del perfil del usuario y le permite realizar 3 acciones, editar su perfil, eliminar todos los movimientos o eliminar el usuario entero para empezar desde 0.

CashFlow no es una aplicación perfecta, es por ello que presenta unas ventajas e inconvenientes:

- Ventajas: es una app muy simple y sencillo apto para cualquier persona y no se necesita iniciar sesión ni una contraseña ya que todo es local. Es una aplicación independiente, no necesita la conexión de ningún tercero.
- Inconvenientes: al ser una app sencilla es verdad que carece de funcionalidades adicionales que podrían ser muy útiles para el usuario y al estar todo guardado en local, si se cambia de dispositivo o se quiere acceder desde otro no es posible.

CashFlow ha tenido un desarrollo bastante complejo en algunos apartados, por ejemplo en la implementación de gráficas. Para ello se ha utilizado un paquete NuGet de DevExpress y se ha tenido que consultar en la documentación de su página web para poder conseguir las gráficas. Otra dificultad fue la encriptación, el problema principal era la clave privada necesaria para desencriptar, al principio se estableció como clave estática en la clase RegisterScreen, pero esto daba error cuando había un usuario ya creado y se quería acceder directamente a la aplicación. Es por ello que se ha finalizado guardando las claves para desencriptar del nombre y apellidos como dos campos de la tabla User.

En resumen, se ha adquirido mucho aprendizaje durante el desarrollo de CashFlow gracias a la afrontación de varios problemas y el resultado final es bastante bueno para un proyecto de alrededor de 30 horas.

5. Bibliografías y referencias

Se ha consultado en muchos sitios para realizar este proyecto, ya sean vídeos de YouTube, documentaciones de Microsoft u otras páginas y repositorios de GitHub de otros autores con ejemplos.

- App Development. (s. f.). TECHCOMMUNITY.MICROSOFT.COM.
<https://techcommunity.microsoft.com/t5/app-development/bd-p/app-dev>
- Davidbritch. (s. f.). Documentación de la interfaz de usuario de la aplicación multiplataforma de .NET - .NET MAUI. Microsoft Learn.
<https://learn.microsoft.com/es-es/dotnet/maui/>
- Davidbritch. (2023, 5 mayo). Manifiesto de aplicación de Android - .NET MAUI. Microsoft Learn. <https://learn.microsoft.com/es-es/dotnet/maui/android/manifest>
- DevExpress-Examples. (s. f.). GitHub - DevExpress-Examples/maui-charts: This repository contains projects that allow you to create different charts. GitHub.
<https://github.com/DevExpress-Examples/maui-charts/tree/22.2.3%2B>
- Dotnet-Bot. (s. f.). Microsoft.Maui.Controls Namespace. Microsoft Learn.
<https://learn.microsoft.com/en-us/dotnet/api/microsoft.maui.controls?view=net-maui-7.0>
- El Camino Dev. (2022, 4 junio). Curso de .NET MAUI gratis # 1 - Introducción [Vídeo]. YouTube. https://www.youtube.com/watch?v=SwE3n9_9Ybo
- Get Started with Cartesian Chart for .NET MAUI | .NET Multi-platform App UI | DevExpress Documentation. (2023, 18 enero).
<https://docs.devexpress.com/MAUI/403298/charts/get-started-with-cartesian-chart>
- how to navigate in OnAppearing() method? (s. f.). Stack Overflow.
<https://stackoverflow.com/questions/72373278/how-to-navigate-in-onappearing-method>
- .NET MAUI .csproj MSBUILD. (s. f.). Stack Overflow.
<https://stackoverflow.com/questions/75079545/net-maui-csproj-msbuild>
- Programming With Pragnesh. (2022, 15 junio). Performing CRUD Operation In .NET MAUI (SQLite) [Vídeo]. YouTube. <https://www.youtube.com/watch?v=JRNwjsywrWM>

Link APK de CashFlow:

https://drive.google.com/drive/folders/1_7orSsIKCmh_h5LCScS2LfAYRNyfHKuc?usp=sharing