

Aplicación Desarrollada para la empresa
SigmaPyme S.L . Fase Alpha

Proyecto Desarrollo de Aplicaciones Multiplatafor ma

Xamarin

Sergio Delgado Barrios

RESUMEN

El objetivo principal de este proyecto es la creación de una aplicación en xamarin forms para la gestión de la empresa donde he estado de prácticas facilitándoles un poco las labores del día a día.

Se desea proporcionar las herramientas necesarias para la administración de la aplicación y además desarrollar ambas partes utilizando las últimas tecnologías, librerías y patrones de diseño disponible de manera que estos aporten el máximo valor posible tanto a usuarios finales, como a administradores.

Abstract

The main objective of this project is the creation of an application in xamarin forms for the management of the company where I have been in practice facilitating a bit the work of day to day.

It is desired to provide the necessary tools for the administration of the application and also to develop both parts using the latest technologies, libraries and design patterns available so that these provide the maximum possible value to both end users and administrators.

Contenido

Introducción	6
1.1 Contexto y motivación del proyecto	7
1.2 Descripción del proyecto	7
1.3 Objetivos del proyecto	7
1.4 Características Principales del sistema	8
1.5 Estructura de la memoria	8
1.6 Marco tecnológico	8
1.7 Aplicaciones Nativas	9
1.7.1 Android	9
1.7.2 iOS.....	10
1.7.3 Windows	11
1.8 APLICACIONES HÍBRIDAS	13
1.9 APLICACIONES GENERADAS.....	15
1.10 SOLUCIÓN ESCOGIDA Y JUSTIFICACIÓN	16
1.10.1 XAMARIN.....	17
1.10.2 XAMARIN.FORMS	17
1.11 Planificación del proyecto.....	20
ANALISIS	22
2.1 ACTORES DEL SISTEMA	23
2.2 REQUISITOS DE USUARIO	23
2.3 DIAGRAMA DE CASOS DE USO	24
2.3.1 Especificación de los casos de uso.	26
DISEÑO	31
3.1 Selección del entorno de desarrollo.....	32
3.2 Selección de la Base de datos	32
3.2.1 Modelo Lógico de datos.....	34
3.3 Diagrama de Actividad.....	35
3.4 Diseño de la interfaz.....	35
IMPLEMENTACIÓN.....	47
4.1 Herramientas empleadas.....	48
4.2 Requisitos Hardware y Software	48
4.2.1 El modelo	48

4.2.2 La vista	49
4.2.3 Modelo de vista (ViewModel)	50
4.3 Acceso a Base de datos.....	51
4.4 Estructura del proyecto	56
Pruebas.....	60
5.1 Prueba de caja blanca.....	61
5.2 Pruebas de caja negra	61
Conclusiones.....	65
6.1 Conclusión.....	66
6.2 Mejoras Futuras	66

Tabla de Ilustraciones

Ilustración 1 Universal Windows Platform	12
Ilustración 2 Comportamiento y desarrollo de aplicaciones nativas	15
Ilustración 3 Ejemplo Xaml	18
Ilustración 4 PCL or Shared project.....	19
Ilustración 5 código compartido	20
Ilustración 6 librerías Xamarin.....	20
Ilustración 7 trello	21
Ilustración 8 Actor - 1	23
Ilustración 9 Actor -2.....	23
Ilustración 10 Diagrama de caso de uso.....	25
Ilustración 11 caso de uso - 1	26
Ilustración 12 caso de uso - 2	27
Ilustración 13 caso de uso -3	28
Ilustración 14 caso de uso - 4	28
Ilustración 15 caso de uso - 5	29
Ilustración 16 caso de uso - 6	29
Ilustración 17 Imagen de xamarin	32
Ilustración 18 Imagen de Mysql.....	33
Ilustración 19 Esquema Entidad-Relación	34
Ilustración 20 Apis más utilizadas (Android)	36
Ilustración 21 colors.xml	36
Ilustración 22 Explicacion Diseño App	37
Ilustración 23 Diseño Inicio Sesión	38
Ilustración 24 Diseño Página Principal	39
Ilustración 25 Diseño Gasolinera	40
Ilustración 26 Diseño Objetivos	41
Ilustración 27 Diseño Detalles Objetivos.....	42
Ilustración 28 Diseño crear Objetivos	43
Ilustración 29 Diseño objetivos Cumplidos	44
Ilustración 30 Diseño Objetivos Ajenos.....	44

Ilustración 31 Diseño Objetivos Ajenos Descripción.....	45
Ilustración 32 Requisitos Hardware y Software	48
Ilustración 33 Ejemplo Model.....	49
Ilustración 34 Ejemplo View	49
Ilustración 35 Ejemplo viewModel	50
Ilustración 36 Imagen MVVM	50
Ilustración 37 Carpetas Servicio Rest	51
Ilustración 38 Dependencias proyecto.....	56
Ilustración 39 Carpeta Servicio Rest proyecto.....	57
Ilustración 40 carpetas de vistas proyecto	59
Ilustración 41 tabla pruebas - Inicio Sesión.....	62
Ilustración 42 tabla pruebas - Truncar Inicio Sesión.....	62
Ilustración 43 tabla pruebas - Visualizar Objetivos.....	63
Ilustración 44 tabla pruebas - crear objetivos	63
Ilustración 45 tabla pruebas - formulario gasolinera.....	63
Ilustración 46 tabla pruebas - visualizar objetivos ajenos.....	64
Ilustración 47 tabla pruebas visualizar objetivos completados.....	64

Introducción

1.1 Contexto y motivación del proyecto

SigmaPyme es una pequeña empresa que se dedica a proporcionar servicios y soluciones informáticos. Ofrece todo tipo de ventajas a pequeñas y medianas empresas, ya sea para informatizar sus instalaciones, incorporar nuevos elementos informáticos o simplemente necesitan soporte relacionado con la informática.

Este proyecto surge de la necesidad que tienen los empleados de SigmaPyme de llevar un control de sus objetivos manteniendo a todos los implicados informados de lo que deben hacer diariamente. Hemos decidido hacer esto a través de un proyecto. Se ha optado por una aplicación multiplataforma para Android e IOS que pueda ser usada tanto por los administradores como por los comerciales. La aplicación móvil muestra los objetivos en tiempo real que deberán de realizar los comerciales, ver los que están ya realizados, poder introducir el importe de la gasolina y enviar una imagen con su recibo a la administrativa.

1.2 Descripción del proyecto

El proyecto consta en crear una aplicación móvil además de crear su API para gestionar la base de datos desde la misma.

Como trabajo en la estancia, el alumno debe determinar los requisitos del proyecto, establecer una estimación de los costes y de recursos y mantener un seguimiento adecuado en cada momento, además de desarrollar la aplicación.

La aplicación debe permitir al usuario iniciar sesión. Si se trata de un comercial esté podrá ver los objetivos tanto cumplidos como no cumplidos y acceder a un formulario para enviar los importes de la gasolina con su foto del ticket. Dicho esto, está claro que hay dos tipos de usuarios: Comerciales y Administradores. El usuario Administrador podrá realizar todo lo citado anteriormente para los comerciales y además tendrá la opción de crear objetivos y de poder ver todos los objetivos cumplidos y no cumplidos de cada usuario.

1.3 Objetivos del proyecto

El objetivo de este proyecto es la creación de una aplicación móvil para Android e IOS con las herramientas de Xamarin, para acceder a la base de datos vamos a tener que crear un servicio REST y una base de datos relacional.

Se espera que los usuarios puedan iniciar Sesión en su cuenta desde la aplicación móvil con las credenciales proporcionadas por el administrador, que muestre información sobre los objetivos,

permita al administrador crear nuevos objetivos, consultarlos y poder ver objetivos realizados y no realizados de sus comerciales además de un formulario donde se inserte el importe que se ha echado de gasolina y enviar una foto con la factura a la administrativa.

1.4 Características Principales del sistema

El producto Desarrollado resultante del estudio adjunto se trata de una aplicación móvil llamada SigmaApp que da soporte a cualquier versión del sistema operativo de Android, aunque ofreciendo a partir de su versión 5.0 una experiencia más completa.

Para satisfacer estos objetivos principales la aplicación deberá de ofrecer las siguientes características:

- ☑ Permitir a los usuarios de la aplicación Iniciar Sesión.
- ☑ Permitir a los usuarios de la aplicación ver sus objetivos no completados.
- ☑ Permitir a los usuarios de la aplicación ver sus objetivos completados.
- ☑ Permitir a los Administradores de la aplicación ver los objetivos completados del resto de usuarios.
- ☑ Permitir a los Administradores de la aplicación ver los objetivos No completados del resto de usuarios.
- ☑ Permitir a los usuarios de la aplicación rellenar el formulario de la gasolinera y enviar la foto del ticket.
- ☑ Permitir a los administradores gestionar de manera sencilla la base de datos creando objetivos.

1.5 Estructura de la memoria

La memoria se estructura en seis capítulos. En el capítulo 1 se expone el contexto y motivación del proyecto, describiendo, grosso modo, la empresa y la necesidad que llevo a proponer este proyecto, los objetivos del proyecto, así como las expectativas del mismo. En el capítulo 2 se describe en detalle el proyecto. El capítulo 3 se centra en la planificación del proyecto, incluyendo la metodología usada, la estimación de recursos y costes del proyecto y el seguimiento del mismo. El análisis y el diseño del sistema se detallan en el capítulo 4, mientras que en el capítulo 5 se describe la implementación y las pruebas hechas. Finalmente, en el capítulo 6 aparecen las conclusiones personales.

1.6 Marco tecnológico

Como parte del inicio del proyecto se realizó un estudio sobre las diferentes opciones disponibles para desarrollar el proyecto. Desde diferentes entornos de desarrollo, librerías externas, servicios, últimas tendencias en desarrollo Android etc... Este capítulo pretende ser un resumen de todo ello.

Existen 4 tipos de aplicaciones para su utilización en Smartphone y Tablet como son:

- Aplicaciones nativas.
- Aplicaciones híbridas.

- Aplicaciones generadas.

- No me voy a centrar en aplicaciones web dado que no lo estudio en mi ciclo.

1.7 Aplicaciones Nativas

Estas aplicaciones nativas están únicamente desarrolladas con el software que ofrece cada sistema operativo a los programadores, generalmente llamado SDK (Software Development Kit, kit de desarrollo software). En este concepto es el que radica la mayor y más significativa diferencia, cada sistema operativo tiene su propio SDK, lo que

conlleva un desarrollo independiente para cada plataforma, es decir, cada aplicación es programada para un sistema operativo específico.

En términos de rendimiento son superiores a cualquier tipo de aplicación, el utilizar directamente el software específico de cada plataforma hace que pueda expresarse al máximo sus capacidades sin necesitar de estar conectados a ningún servidor externo ni hacer uso de la conexión a internet en ninguno de los casos, a no ser que la propia aplicación lo necesite. La apariencia también será un punto a favor, al utilizar los elementos propios de cada sistema operativo las aplicaciones tendrán una mayor coherencia y consistencia con el resto de las aplicaciones del dispositivo (Cuello et al, 2013).

Un gran inconveniente a tener en cuenta en el caso de pretender crear aplicaciones multiplataforma es el que radica en la primera característica que se ha mencionado, un desarrollo independiente por cada plataforma. Si por ejemplo estamos buscando crear una aplicación para los tres sistemas operativos más demandados las cuentas serán fáciles, deberemos invertir aproximadamente el triple de tiempo que para una aplicación en una única plataforma. Si este es el desarrollo que se desea, se aportarán unos breves conocimientos acerca de los requisitos de cada plataforma.

1.7.1 Android

Se trata de un sistema operativo basado en Linux siguiendo un modelo de código abierto. Es el sistema operativo líder en el mercado de Smartphones y Tablets.

Algunas de sus ventajas y desventajas son las siguientes.

Ventajas:

- Modelo de código abierto, cualquier persona puede realizar una aplicación para Android.
- Lenguaje de programación JAVA, ampliamente extendido, únicamente será necesario instalar su JDK (Java Development Kit, kit de desarrollo Java) el cual es posible instalarlo junto al entorno de desarrollo propio de Google, Android Studio. Dando la posibilidad de que cualquier usuario pueda desarrollar sus aplicaciones y mediante una licencia muy accesible pueda añadirlas a la tienda de aplicaciones.

Desventajas:

- El gran número de dispositivos y de variantes de sistema operativo hace que pocos terminales reciban soporte durante un largo periodo de tiempo.
- La posibilidad de multitarea conlleva un alto consumo de batería.

1.7.2 iOS

Sistema operativo propiedad de Apple Inc. Originalmente desarrollado para el primer iPhone por ello su antecesor se llamaba iPhone OS, posteriormente adaptado para el iPod touch y el iPad.

Para desarrollar aplicaciones en este sistema operativo es indispensable trabajar con el IDE Xcode dentro de una máquina con sistema operativo MacOS, por lo que ya se impone la obligación de contar con un equipo ya sea sobremesa o portátil de Apple. Se tienen dos posibilidades de lenguaje con los que trabajar:

- Objective-C, extensión orientada a objetos del lenguaje C.
- Swift, más amigable que Objective-C, permite trabajar junto a este.

Una de sus desventajas de cara al desarrollo para iOS es la obligación de contar con una cuenta de desarrollador iOS de Apple, con ella podremos acceder a todas sus ventajas a través de una suscripción de pago anual, desde proyectos de ejemplo hasta manuales de ayuda a parte de poder unirnos a la gran comunidad de desarrolladores de Apple (Arias et al, 2016).

Al igual que con todos los sistemas operativos, este tendrá sus ventajas e inconvenientes. Aquí veremos algunas de sus ventajas:

- Al estar diseñado para unos dispositivos en concreto, la optimización de código es máxima.
- La gran mayoría de los dispositivos durante su ciclo de vida se mantienen completamente actualizados.
- Seguridad y exigencia en su proceso de publicación de aplicaciones a través de la App Store.

Algunas de sus desventajas:

- Utiliza lenguajes de programación menos conocidos.
- Necesidad de una cuenta con suscripción anual de pago para su despliegue.
- Poca libertad en cuanto a la personalización de aplicaciones.

1.7.3 Windows

Si se desea un máximo despliegue hacia la mayor cantidad de equipos Windows, la opción más interesante es trabajar con una versión concreta y limitada de Windows como es la nueva plataforma universal de Windows (UWP, Universal Windows Platform), siendo una plataforma compatible con la totalidad de dispositivos que ejecutan Windows 10 sin importar si se trata de equipos más o menos complejos, desde un equipo de sobremesa hasta una Surface. Permitiendo que un único paquete de instalación sea compatible con la totalidad de los equipos que utilicen este sistema. Como se ve en la siguiente ilustración, donde tenemos la totalidad de familias compatibles con este nuevo sistema de aplicaciones.



Ilustración 1 Universal Windows Platform

Es posible que existan problemas de compatibilidad con algunos paquetes que si estarían disponibles para una aplicación común de escritorio, pero gracias a una gran comunidad tras esta plataforma es posible instalar plugins compatibles que realicen la misma tarea de forma transparente para el usuario (Microsoft, 2018).

Algunas de sus ventajas son las siguientes:

- Instalación más rápida y sencilla.
- Utilizan un centro de actualizaciones integrado para estas Apps.
- Aprovechan todas las funcionalidades que proporcione el equipo en el que se ejecutan.
- Gracias a su funcionamiento interno, la seguridad es mayor que en las aplicaciones comunes de escritorio.

También tienen sus desventajas:

- Falta de despliegue de Windows 10.
- No existe retro compatibilidad con versiones anteriores del sistema operativo, es posible acceder a una aplicación UWP de forma remota desde sistemas operativos anteriores, pero se resentiría el rendimiento.

1.8 APLICACIONES HÍBRIDAS

Este tipo de aplicaciones son una combinación de los dos tipos anteriores que se han visto. Su desarrollo es similar al de las aplicaciones web. La única diferencia consiste en que una vez finalizada la aplicación web, debe empaquetarse como si de una aplicación nativa se tratase. Mediante un mismo código desarrollado para navegador es posible obtener una aplicación nativa y única por cada plataforma y de este modo realizar una distribución a través de las diferentes tiendas de aplicaciones.

Para el usuario final da la impresión de un trabajo más logrado, bien es cierto que sigue sin utilizar los elementos característicos del sistema operativo en el que se utiliza, pero permite trabajar con algunas de las capacidades internas que en el caso de trabajar desde un navegador web no sería posible. Este acceso a las capacidades internas marcará la diferencia entre los sistemas operativos a los que llegará, ya que el código utilizado en cada plataforma será único, y será el desarrollo a mayores que necesitará comparado con una web App. Sin olvidar la mejora en cuanto a rendimiento si se compara trabajar a través de un navegador con trabajar desde la propia aplicación, siempre que la complejidad no sea excesiva.

En términos de mantenimiento tiene las ventajas de una aplicación web, salvo en lo referente al acceso a las capacidades internas del dispositivo, siendo únicamente necesario mantener un único código para todas las plataformas añadiendo el trabajo de actualizar la aplicación en las respectivas tiendas.

Entrando con detalle en su desarrollo existen diferentes frameworks que permiten el entrar de lleno en la producción de aplicaciones híbridas. PhoneGap y Apache Cordova son herramientas muy similares que permiten este tipo de desarrollos.

Ambas opciones son gratuitas, de código abierto y se utilizan del mismo modo. PhoneGap, propiedad de Adobe es una distribución de Apache Cordova. Con lo que los servicios que ofrecen son similares entre ambas herramientas. El único añadido es el servicio de compilación para PhoneGap que proporciona Adobe, haciendo más simple su trabajo sin ser necesaria la instalación del SDK para su compilación. En el caso de proyectos de código abierto este servicio es gratuito, todo lo contrario que si se trabaja en una aplicación comercial.

Centrándonos exclusivamente en las ventajas e inconveniente de una aplicación híbrida tenemos las siguientes :

Ventajas:

- Acceso a la gran mayoría de las capacidades del dispositivo.
- Mantenimiento único para el código compatible a todas las plataformas.
- Conectividad online y offline.
- Menor coste de desarrollo en aplicaciones multiplataforma.

Desventajas:

- A pesar de tener un buen rendimiento comparado con una aplicación web, nunca llegaremos al rendimiento de las aplicaciones nativas. En el caso de que la aplicación sea una aplicación muy compleja ello afectará notablemente al rendimiento.
- Las tiendas de aplicaciones ponen bastantes restricciones al desarrollo de aplicaciones híbridas, siendo beneficiosas de cara a las aplicaciones nativas.

1.9 APLICACIONES GENERADAS

Se trata de una tecnología relativamente nueva, aun no muy extendida con algunas compañías apostando fuerte por este tipo de desarrollo. Sus aplicaciones son desarrolladas con un lenguaje de programación específico del software utilizado, una vez desarrollado y a través de este código, se genera un código nativo independiente para cada plataforma a desarrollar. Comparten con las híbridas el hecho de reciclar código, a pesar de ello, no se parecen en nada a las aplicaciones híbridas, ya que, en el caso de las aplicaciones generadas, este código estándar se adapta a cada sistema operativo concreto.

El comportamiento y desarrollo de las aplicaciones nativas sería el que se ve en la siguiente ilustración:

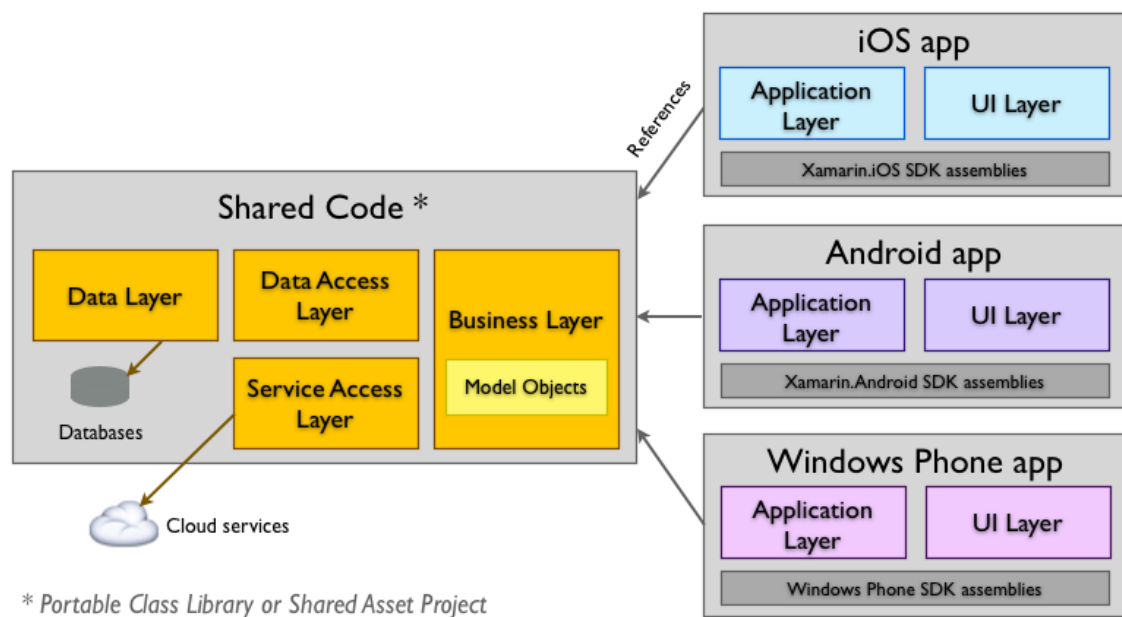


Ilustración 2 Comportamiento y desarrollo de aplicaciones nativas

A través de un código compartido el compilador es el encargado de generar el código nativo para Android, iOS o Windows.

A pesar de trabajar con un código independiente al de cada plataforma en concreto, es importante el conocer algo de cada lenguaje específico. El hecho de desarrollar una aplicación multiplataforma supone un coste claramente menor al de las aplicaciones nativas, haciendo que el rendimiento sea prácticamente similar, lo que puede hacer muy rentable decidir utilizar esta tecnología antes que aplicaciones web o híbridas, con un pequeño coste a mayores el rendimiento es claramente superior.

Para la parte de pruebas, emulación y simulación de las aplicaciones generadas el software existente es extremadamente exigente de momento. Para iOS será indispensable el contar con un equipo macOS en la misma red que el equipo de desarrollo, a fin de poder compilar y probar a través de Xcode la aplicación o bien en uno de sus emuladores de iPhone o iPad, o bien en un dispositivo físico conectado por cable al equipo macOS. Para Windows debe ser compilado en un PC con Windows 10 y este mismo equipo trae consigo los diferentes simuladores de

dispositivos necesarios. El caso de Android es el más especial basta con tener instalado el SDK correspondiente de Android y el mismo software trae consigo los emuladores necesarios para hacer pruebas, o si se desea teniendo el Smartphone o Tablet en modo depuración, es posible compilar y ejecutar paso a paso las aplicaciones directamente en el dispositivo.

1.10 SOLUCIÓN ESCOGIDA Y JUSTIFICACIÓN

Tras un recorrido a través de todas las posibles tecnologías para el desarrollo de la aplicación ahora es el momento de la decisión más importante de todas, la elección del software correcto que permita el desarrollo de esta aplicación para el tratamiento de pacientes en cuanto a la evaluación y rehabilitación de las capacidades cognitivas. Para ello es importante ir haciéndose una serie de preguntas a fin cerrar el cerco lo más posible hacia la tecnología más conveniente.

no puedo desarrollar esta aplicación como tres aplicaciones nativas independientes.

Me quedan dos posibles opciones, una aplicación híbrida o una aplicación generada. Ambas con algunas similitudes y algunas diferencias. A pesar de tener en ambas un código común, las aplicaciones generadas requieren de algo más de conocimiento en cuanto a cada plataforma de manera independiente. Esto también afecta a su rendimiento, haciendo que su rendimiento sea muy próximo al de una aplicación nativa. Ahora se debe pensar, ¿merece la pena ese coste mayor en materia de investigación en favor de una mejora significativa de rendimiento? La respuesta claramente es sí.

La mejor opción disponible para el tipo de desarrollo que se necesita es la de desarrollar una aplicación generada. Aquí es donde buscamos el equilibrio entre coste y rendimiento, no llegaremos al rendimiento perfecto que daría como resultado una aplicación nativa al 100%, pero sí que nos aproximamos enormemente a este con un coste mucho menor.

Como herramienta de trabajo se ha decidido utilizar Visual Studio 2019, mediante la API (Application Programming Interfaces, Interfaces de programación de aplicaciones) de Xamarin. A continuación, se presenta los motivos por los que se ha escogido esta herramienta:

- Utiliza un lenguaje fácil de manejar como es C#.
- Posibilidad de crear soluciones multiplataforma y desplegarlas rápidamente.
- La gran comunidad de Microsoft que hay detrás permite encontrar una amplia cantidad de manuales de ayuda para su desarrollo.
- Software libre para su desarrollo multiplataforma, el software Xamarin se incluye de forma gratuita con la versión Community de Visual Studio.

Para proseguir se va a entrar con más detalle en todas y cada una las características de Xamarin.

1.10.1 XAMARIN

Xamarin es un entorno de desarrollo, el cual, permite crear aplicaciones multiplataforma para Android, iOS y Windows utilizando un lenguaje de programación común, en este caso C#. No es necesario aprender el lenguaje de una plataforma específica, es decir, facilita la vida al programador ya que este puede trabajar únicamente conociendo las características propias de xamarin y de este lenguaje de programación. Existen varios frameworks con Xamarin integrado, sin embargo, el más recomendado es visual studio, propiedad de Microsoft.

Xamarin permite combinar el rendimiento logrado en aplicaciones nativas con unas características eficaces (Xamarin Microsoft, 2019):

- Xamarin contiene enlaces para casi todos los SDK de las plataformas de iOS y Android, siendo compatible con diferentes versiones produciendo menos errores en tiempo de ejecución.
- Interoperabilidad con bibliotecas específicas de cada plataforma, permitiendo invocar las bibliotecas de Objective-C, Java, C y C++. Permite utilizar código de terceros en nuestras aplicaciones para el código de una plataforma en concreto.
- Compatibilidad multiplataforma móvil: Ofrece compatibilidad con las tres principales plataformas móviles, con una API unificada que permite acceder a los recursos de las tres plataformas. permitiendo reducir el tiempo de desarrollo de las aplicaciones.

Incluidas en Xamarin existen diferentes herramientas para facilitar tanto el desarrollo como el diseño de la aplicación, la más conocida de ellas es Xamarin.Forms.

1.10.2 XAMARIN.FORMS

Se trata de un kit de herramientas de interfaz de usuario multiplataforma, permitiendo crear diseños semejantes a los de las aplicaciones nativas, con una diferencia, los elementos se comparten entre las aplicaciones de todas las plataformas cambiando según en la que se ejecute. Existen unas limitaciones en cuanto al sistema operativo de la plataforma destino (Xamarin Forms, 2019):

Estas aplicaciones se pueden desarrollar tanto en equipos Windows como macOS, sin embargo es obligatorio su uso en Windows junto con Visual Studio para el desarrollo de su versión en Windows, si junto a estas versiones se desea poder compilar y ejecutar en iOS será necesario disponer de un equipo MAC en la misma red.

Al poderse considerar aplicaciones nativas las generadas mediante Xamarin.Forms, no tienen las limitaciones de otras aplicaciones multiplataforma en cuanto al espacio aislado, el acceso limitado a APIs o su bajo rendimiento. Estas aplicaciones pueden utilizar cualquier API así, acceder a cualquier recurso del dispositivo. No es necesario crear toda la interfaz gráfica de la aplicación mediante Xamarin.Forms, es posible crear partes mediante el kit de herramientas nativo y diseñar esta combinando ambas.

Tenemos dos técnicas para la creación de interfaces gráficas al igual que sucede con las aplicaciones nativas:

- Mediante código fuente C#: La interfaz gráfica se genera junto al código ejecutable, como sucede por ejemplo en Android la creación de interfaz gráfica a través de código es menos recomendable, supone un mayor consumo para el procesador.

- Uso de XAML (eXtensible Application Markup Language, Lenguaje Extensible de Formato para Aplicaciones): Lenguaje especialmente diseñado para la creación de la interfaz de usuario base en la presentación de Windows. Se basa en XML (eXtensible Markup Language, Lenguaje de Marcado Extensible) optimizado para su uso en interfaces gráficas complejas haciéndolo más sencillo e intuitivo. La mayor ventaja es que el diseño a través de XAML no requiere de ningún tipo de programación siendo relativamente sencillo su diseño si se conoce ligeramente la estructura de etiquetas y propiedades.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="SergioDelgadoProyecto.formObjetivos.Objetivos">
    <ContentPage.Content>
        <StackLayout>
            <ListView x:Name="MyListView" ItemSelected="OnItemSelected">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <TextCell Text="{Binding objetivos}" TextColor="Black"/>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Ilustración 3 Ejemplo Xaml

De este modo es posible generar una misma interfaz gráfica de dos métodos completamente diferentes, y como vemos en la ilustración el resultado es el mismo con la diferencia de la apariencia de los elementos como si de elementos nativos se tratase.

En cuanto a la arquitectura de Xamarin.Forms se basa en la maximización del código a compartir entre plataformas. Respetando los principios de programación orientada a objetos para obtener una aplicación con capas diferenciadas e independientes:

- Encapsulación: garantizar que las capas y clases llevan a cabo sus funciones ocultando los detalles de su implementación. No es necesario saber cómo realizan las tareas de forma interna los objetos.
- Separación de responsabilidades: cada componente debe realizar únicamente sus funciones y exponer sus funcionalidades a través de una API accesible para los demás.

- Polimorfismo: programación a una interfaz compartida entre plataformas la cual interactúa con características específicas de la plataforma.

A nivel de arquitectura existen dos tipos de proyectos en función de la librería de clases que se utilice, podemos estar ante un proyecto compartido (Shared Project) o bien en un proyecto con una PCL (Portable Class Library, Librería de clases portátil).

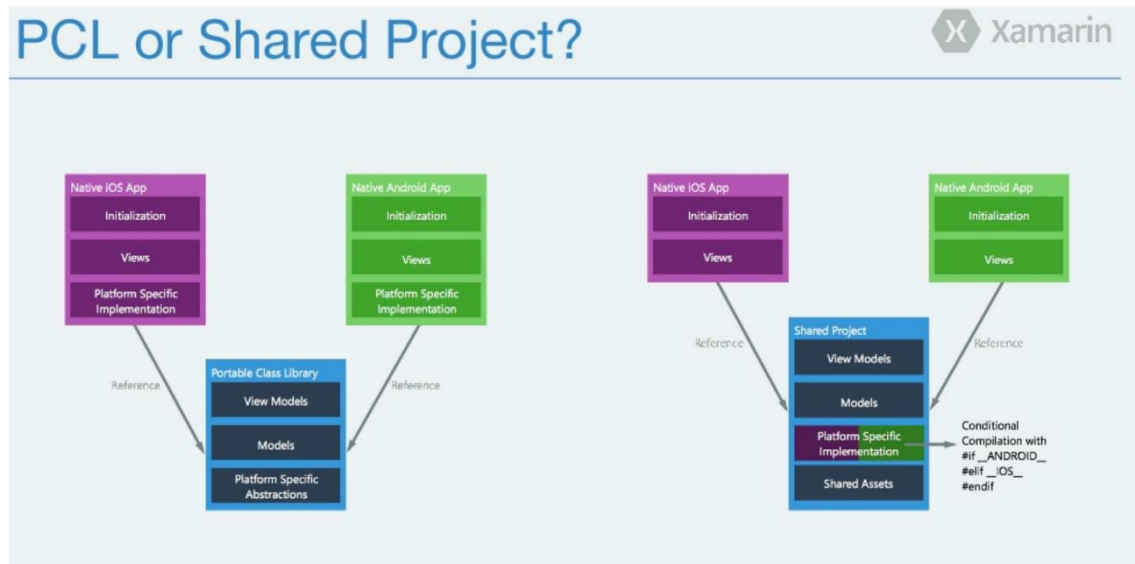


Ilustración 4 PCL or Shared project

- PCL: Para la generación de aplicaciones multiplataforma independientes, es necesario, un proyecto por cada plataforma. A estos tres proyectos se les añade un proyecto PCL compatible con ellas. El código del proyecto PCL se compila contra sus propias referencias, estas librerías son compatibles con Android, iOS y Windows. Creando un fichero .dll para cada plataforma y finalmente pudiendo crear un fichero .APK, .IPA o .APP según el sistema operativo destino.

- Shared Projects: El proyecto compartido no produce ningún .dll, es una agrupación del código que se compila directamente contra las propias referencias de los proyectos independientes de cada sistema operativo destino. Debemos tener referenciadas las mismas librerías en cada proyecto independiente, de no ser así, la compilación devolvería un error en una de las tres plataformas con la notificación de no conocer la referencia que se le pide.

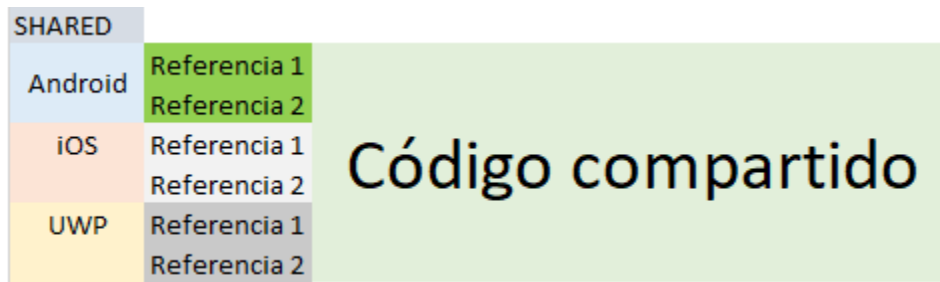


Ilustración 5 código compartido

Al utilizar librerías separadas para cada solución, es posible que no todas tengan un código compatible entre ellas, aquí es donde viene la compilación condicional. La cual nos permite distinguir secciones de código de modo que únicamente se compilen contra las librerías de un sistema operativo concreto, siendo invisibles para los demás.

```
#if __ANDROID__
    Android.Media.MediaPlayer player = new Android.Media.MediaPlayer();
#elif WINDOWS_UWP
    Windows.Media.Playback.MediaPlayer player = new Windows.Media.Playback.MediaPlayer();
#elif __IOS__
    AVFoundation.AVAudioPlayer player;
#endif
```

Ilustración 6 librerías Xamarin

Si se entra en el contexto de la aplicación a desarrollar, se optó por la opción de un proyecto compartido (Shared Project). Para el acceso a las capacidades internas del dispositivo en algunas ocasiones es necesario de utilizar librerías de terceros. Esto provoca que pueda haber problemas de compatibilidad, como así fue. Recurriendo a diferentes bibliotecas para cada solución, siendo necesario utilizar una compilación condicional siempre que hubiera que acceder a estas.

1.11 Planificación del proyecto

La planificación inicial se realizó fijando el inicio del proyecto en marzo. Por ello, se estimó una primera interacción que duraría un mes, en ésta tendría un mayor peso la investigación tras la que se construiría un prototipo muy básico que apenas permitiría la autenticación de usuario para acceder a la aplicación.

La fecha fin del proyecto es el día 10 de junio, por lo tanto, para ese día me he tenido que organizar de la siguiente manera:

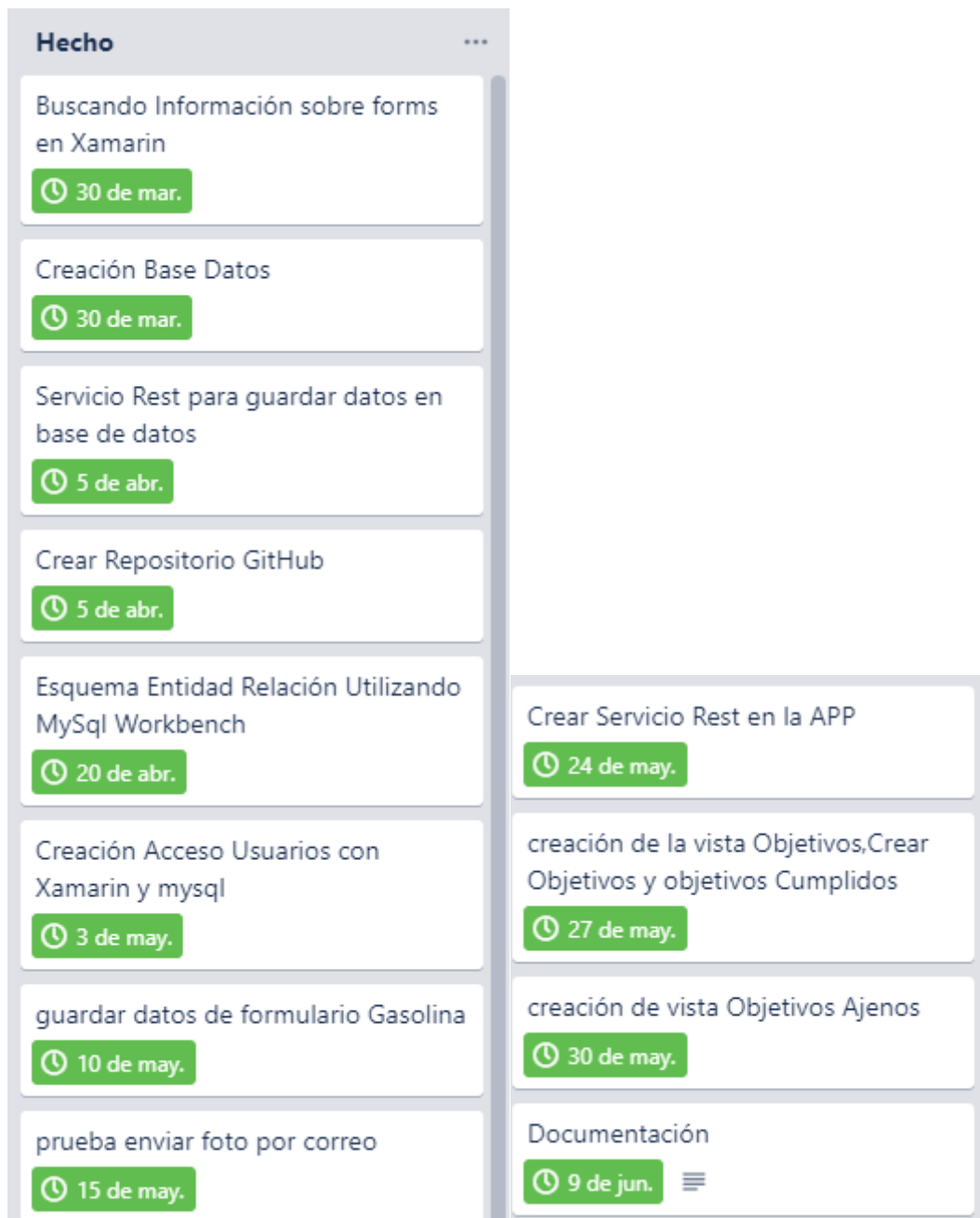


Ilustración 7 trello

Estos han sido los plazos que he ido cumpliendo desde que empecé las practicas hasta la entrega del proyecto comenzando por un largo periodo de documentación y análisis para luego ir implementando todas las ventanas dentro de la aplicación.

ANALISIS

2.1 ACTORES DEL SISTEMA

Se distinguen dos tipos de actores con la posibilidad de interactuar con el sistema:
Administrador y Comercial

ACT-01	Administrador
Versión	1.0
Autor	Sergio Delgado Barrios
Descripción	Este actor representa al Administrador de la aplicación con Acceso a la aplicación y a la creación de usuarios además de poder ver los objetivos de los comerciales.
Comentarios	Debe conocer su contraseña para poder iniciar sesión .

Ilustración 8 Actor - 1

ACT-02	Comercial
Versión	1.0
Autor	Sergio Delgado Barrios
Descripción	Este actor representa al comercial de la aplicación con acceso a esta , solo podrá ver sus objetivos finalizados y no finalizados , ver el formulario de la gasolinera y poder completar objetivos
Comentarios	Debe conocer su contraseña para poder iniciar sesión.

Ilustración 9 Actor -2

2.2 REQUISITOS DE USUARIO

En este apartado se especificarán todos los requisitos de usuario que el sistema debe de cumplir dónde entendemos por requisito de usuario, todos los objetivos o tareas que una clase de usuario podrá realizar con el sistema. Éstos desembocan en casos de uso.

Comerciales

RU-1. Un usuario Comercial podrá autenticarse en la aplicación haciendo uso de sus credenciales de acceso.

RU-2. Un usuario Comercial podrá usar el formulario para la Gasolina

RU-3. Un usuario Comercial podrá Ver y completar sus objetivos.

Administradores

RU-4. Un usuario Administrador podrá Iniciar Sesión.

RU-5 Un usuario Administrador podrá Cerrar Sesión.

RU-6 Un usuario Administrador podrá ver sus objetivos.

RU-7 Un usuario Administrador podrá ver los objetivos de los comerciales (Finalizados y no Finalizados).

RU-8 Un usuario Administrador podrá usar el formulario para la Gasolina.

2.3 DIAGRAMA DE CASOS DE USO

En este apartado se puede ver un diagrama de casos de uso. En este se muestran los actores definidos en el apartado anterior caracterizando las acciones que pueden realizar cada uno de ellos.

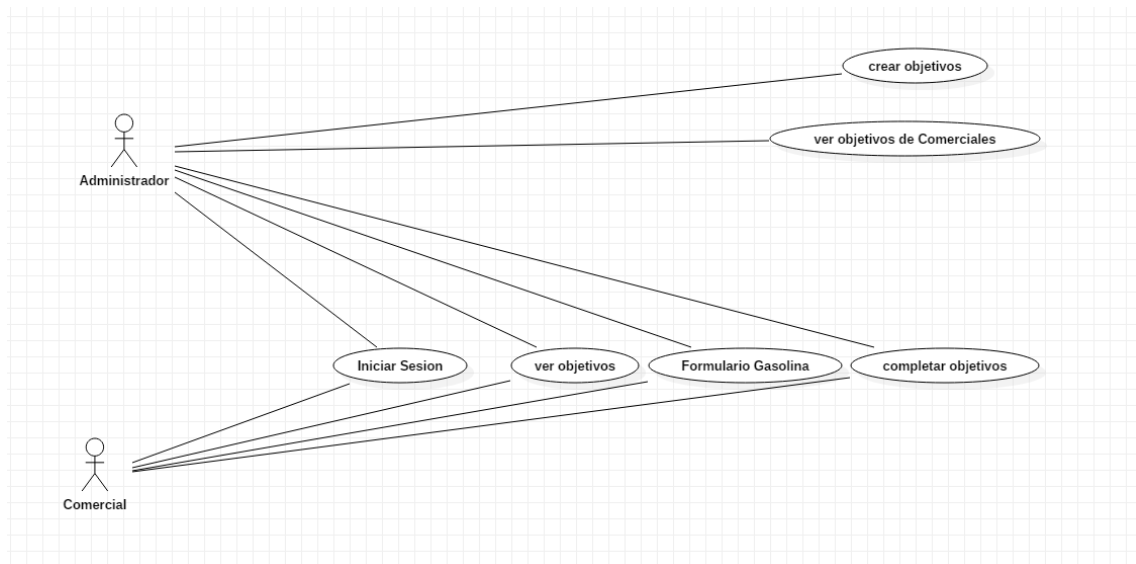


Ilustración 10 Diagrama de caso de uso

2.3.1 Especificación de los casos de uso.

En este apartado se va a especificar cada uno de los casos de uso que han sido plasmados en el diagrama anterior.

CU-01	Iniciar sesión
Versión	1.0
Autor	Sergio Delgado Barrios
Descripción	El sistema permitirá iniciar sesión al usuario con sus credenciales de acceso.
Precondición	Estar registrado en la base de datos
Secuencia normal	<ol style="list-style-type: none">1. Abrir la aplicación2. Rellenar los campos de credenciales3. Pulsar entrar4. El sistema comprueba si existe la cuenta5. El usuario es autenticado y se actualiza la interfaz
Postcondiciones	El usuario se autentica.
Excepciones	<ol style="list-style-type: none">1. Si las credenciales están vacías se indica que el usuario o contraseña no son correctos.2. Si las credenciales están rellenas, pero no coinciden, indica error de autenticación.3. Si no se dispone de conexión a internet, indica error de autenticación.
Frecuencia	Alta
Importancia	Alta

Ilustración 11 caso de uso - 1

CU-02	Ver Objetivos
Versión	1.0
Autor	Sergio Delgado Barrios
Descripción	El usuario podrá ver los objetivos que aún no ha completado
Precondición	Estar dentro de la aplicación
Secuencia normal	<ol style="list-style-type: none"> 6. Abrir Menú lateral 7. Pulsar en ver objetivos 8. Muestra de todos los objetivos 9. Clickar en uno de ellos para verlo con más profundidad 10. Darle a completar si esta completado.
Pos condiciones	El usuario se autentica.
Excepciones	
Frecuencia	Alta
Importancia	Alta

Ilustración 12 caso de uso - 2

CU-03	Formulario Gasolinera
Versión	1.0
Autor	Sergio Delgado Barrios
Descripción	El usuario podrá mandar a la base de datos el importe de la gasolinera y mandar una foto con su ticket a la administrativa por correo.
Precondición	Estar dentro de la aplicación
Secuencia normal	<ol style="list-style-type: none"> 1. Abrir Menú lateral 2. Pulsar en Gasolinera 3. Rellenar formulario 4. Cargar imagen 5. Pulsar en enviar
Pos condiciones	El usuario se autentica.
Excepciones	<ol style="list-style-type: none"> 1. No se pueden enviar caracteres que no sean numéricos. 3. Si no se dispone de conexión a internet, indica error.
Frecuencia	Alta
Importancia	Alta

Ilustración 13 caso de uso -3

CU-04	Completar Objetivos
Versión	1.0
Autor	Sergio Delgado Barrios
Descripción	El usuario puede completar objetivos una vez haya entrado en uno de ellos.
Precondición	Estar dentro de la aplicación y haber clicado en un objetivo en la ventana objetivos
Secuencia normal	<ol style="list-style-type: none"> 1. Abrir Menú lateral 2. Pulsar en ver objetivos 3. Muestra de todos los objetivos 4. Clickar en uno de ellos para verlo con más profundidad 5. Darle a completar si esta completado.
Pos condiciones	El usuario se autentica.
Excepciones	
Frecuencia	Alta
Importancia	Alta

Ilustración 14 caso de uso - 4

CU-05	Crear Objetivos
Versión	1.0
Autor	Sergio Delgado Barrios
Descripción	El usuario Administrador podrá crear objetivos para sus comerciales.
Precondición	Estar dentro de la aplicación
Secuencia normal	<ol style="list-style-type: none"> 1. Abrir Menú lateral 2. Pulsar en crear objetivos 3. Rellenar formulario 4. Pulsar en enviar
Pos condiciones	El usuario se autentica.
Excepciones	1. Si no se dispone de conexión a internet, indica error.
Frecuencia	Alta
Importancia	Alta

Ilustración 15 caso de uso - 5

CU-06	Ver Objetivos Ajenos
Versión	1.0
Autor	Sergio Delgado Barrios
Descripción	El usuario Administrador podrá ver los objetivos completos y no completos de cada usuario.
Precondición	Estar dentro de la aplicación
Secuencia normal	<ol style="list-style-type: none"> 1. Abrir Menú lateral 2. Pulsar en ver objetivos Ajenos 3. Buscar el nombre del usuario al que quiera verle los objetivos. 4. Pulsar el botón.
Pos condiciones	El usuario se autentica.
Excepciones	1. Si no se dispone de conexión a internet, indica error.
Frecuencia	Baja
Importancia	Alta

Ilustración 16 caso de uso - 6

CU-07	Modifica o Borrar Objetivos Ajenos
Versión	1.0
Autor	Sergio Delgado Barrios
Descripción	El usuario Administrador podrá Modificar y borrar los objetivos ya creados de los usuarios.
Precondición	Estar dentro de la aplicación
Secuencia normal	<ol style="list-style-type: none"> 1. Abrir Menú lateral 2. Pulsar en ver objetivos Ajenos 3. Buscar el nombre del usuario al que quiera verle los objetivos. 4. Pulsar el botón. 5. Acceder a un objetivo
Pos condiciones	El usuario se autentica.
Excepciones	1. Si no se dispone de conexión a internet, indica error.
Frecuencia	Baja
Importancia	Alta

DISEÑO

3.1 Selección del entorno de desarrollo

Visual Studio es un conjunto completo de herramientas con la generación de aplicaciones en .NET, Servicios web XML, aplicaciones de escritorio y aplicaciones móviles, visual Basic, visual C# y visual C++ en donde todos utilizan el mismo IDE. A la vez estos lenguajes utilizan funciones de .NET framework que hace más simplificado el desarrollo de aplicaciones web ASP y servicios web XML.

Para optimizar el desarrollo de una aplicación orientada a nivel empresarial he utilizado xamarin. Esta tecnología nos permite crear interfaces de usuarios nativas 100% personalizadas para cada plataforma. Usando forms cuando lo prefiera para optimizar el uso compartido de código, o bien usar los diseñadores nativos para Android e iOS cuando optimice para las interfaces de usuario personalizadas.



Ilustración 17 Imagen de xamarin

3.2 Selección de la Base de datos

Para este proyecto utilizo como base de datos MySQL porque es Open Source además de su velocidad al realizar operaciones y su facilidad de configuración e instalación.

MySQL es un sistema de base de datos relacional, lo que quiere decir que archiva datos en tablas separadas en lugar de guardar todos los datos en un gran archivo, lo que le permite tener mayor velocidad y flexibilidad. Estas tablas están relacionadas de formas definidas, por lo que se hace posible combinar distintos datos en varias tablas y conectarlos.

Algunas características de MySQL son:

Permite escoger múltiples motores de almacenamiento para cada tabla.

Agrupación de transacciones, pudiendo reunir las de forma múltiple desde varias conexiones con el fin de incrementar el número de transacciones por segundo.

Conectividad segura.

Ejecución de transacciones y uso de claves foráneas.

Presenta un amplio subconjunto del lenguaje SQL.

Replicación

Disponible en casi todas las plataformas o sistemas.

Búsqueda de indexación de campos de texto.

Utiliza varias herramientas para portabilidad.

Tablas hash en memorias temporales

Ofrece un sistema de contraseñas y privilegios seguros de verificación basada en el host y tráfico de contraseñas encriptado al conectarse a un servidor.

Soporta gran cantidad de datos, incluso con más de 50 millones de registros.

En las últimas versiones, se permiten hasta 64 índices por tablas. Cada índice puede consistir desde 1 a 16 columnas o partes de columnas. Los máximos anchos de límite son de 1000 bytes.



Ilustración 18 Imagen de Mysql

3.2.1 Modelo Lógico de datos

En este apartado presentamos el modelo entidad-relación de la base de datos realizado con MySQL WorkBench.

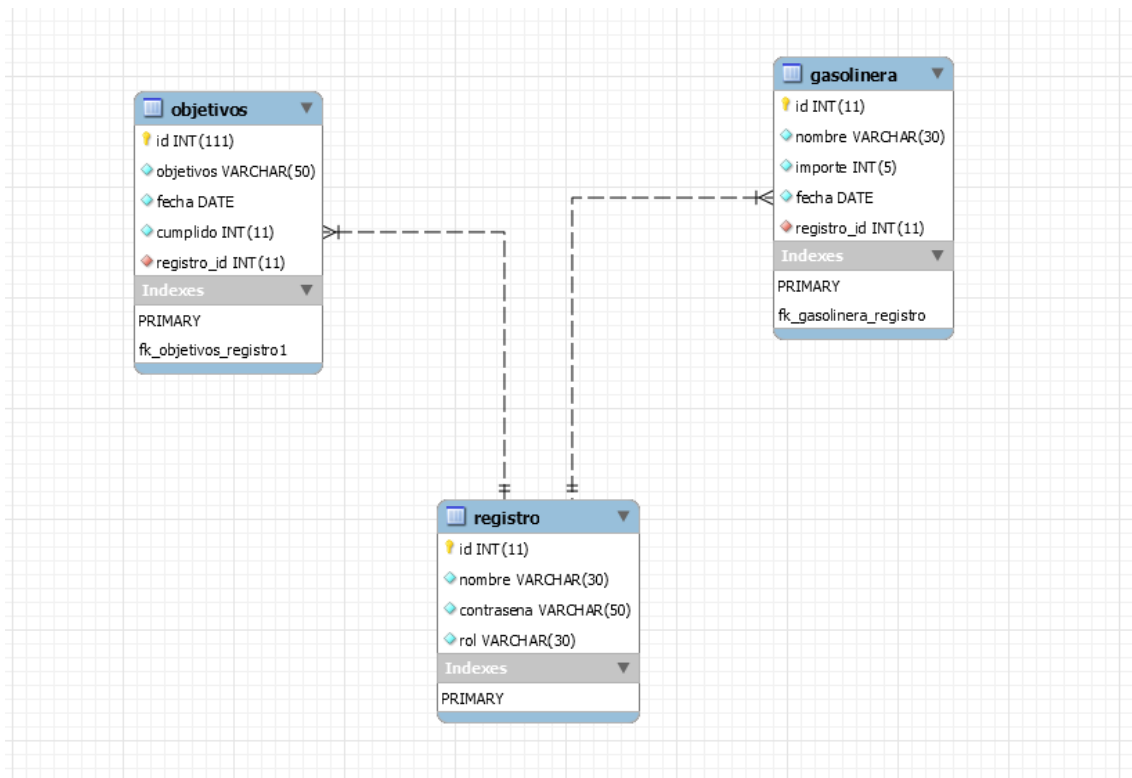


Ilustración 19 Esquema Entidad-Relación

Este sería el esquema Entidad-Relación de nuestra base de datos, como podemos comprobar objetivos y gasolinera tiene una clave foránea de Registro (id).

La Tabla Registro contiene:

Id: identificador de persona.

Nombre: nombre del usuario.

Contraseña: contraseña del usuario encriptada (sha1).

Rol: Rol del usuario.

La tabla Objetivos contiene:

Id: identificador del objetivo.

Objetivos: detalles del objetivo.

Fecha: fecha de creación del objetivo.

Cumplido: si ha sido cumplido el objetivo o no.

Registro_id: id de la persona a la que va destinada el objetivo.

La tabla Gasolinera contiene:

Id: identificador de la gasolina.

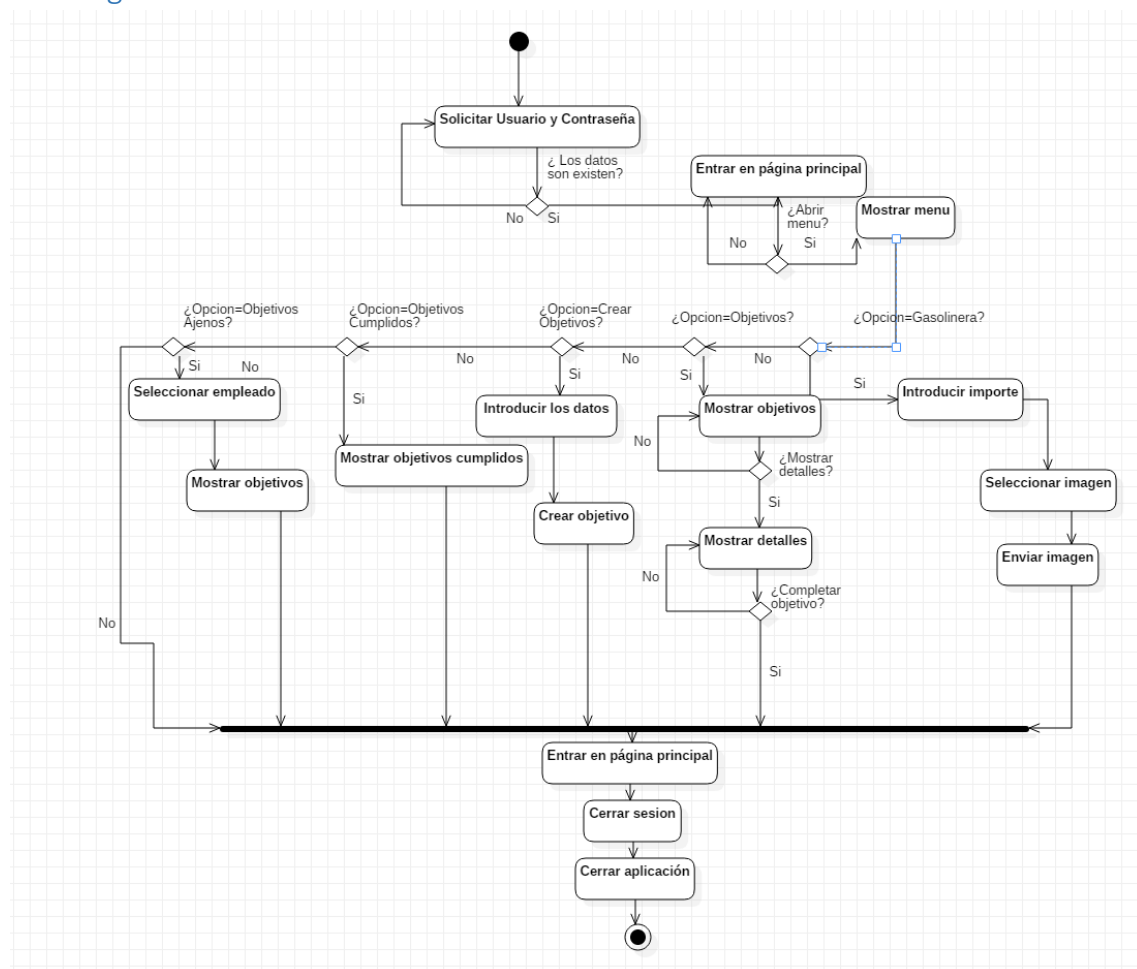
Nombre: nombre del usuario que ha repostado gasolina.

Importe: cantidad de gasolina repostada.

Fecha: fecha de cuando ha repostado gasolina.

Registro_id: id de la persona a la que va destinada el importe de la gasolina.

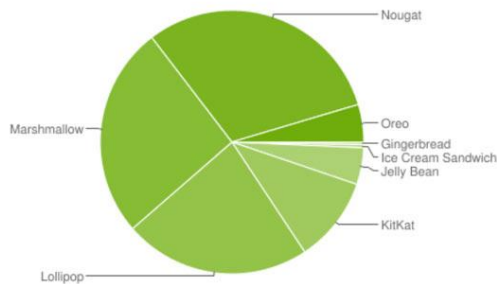
3.3 Diagrama de Actividad.



3.4 Diseño de la interfaz

Al momento de crear nuestro proyecto se selecciona como API mínima la API 23 conocida como “Marshmallow” ya que nos permitiría que nuestra aplicación funcionase en el 100% de los dispositivos según la infografía facilitada por Android Studio.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.4%
4.1.x	Jelly Bean	16	1.7%
4.2.x		17	2.2%
4.3		18	0.6%
4.4	KitKat	19	10.5%
5.0	Lollipop	21	4.9%
5.1		22	18.0%
6.0	Marshmallow	23	26.0%
7.0	Nougat	24	23.0%
7.1		25	7.8%
8.0	Oreo	26	4.1%
8.1		27	0.5%



Data collected during a 7-day period ending on April 16, 2018.
Any versions with less than 0.1% distribution are not shown.

Ilustración 20 Apis más utilizadas (Android)

La primera decisión que se tomó a nivel de diseño fue la selección de una correcta paleta de colores con la que poder trabajar. Para ello, se debía de elegir un color primario, un primario oscuro y un color de acentuación que fueran la base de nuestra paleta de colores.

```
<resources>
    <color name="launcher_background">#FFFFFF</color>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

Ilustración 21 colors.xml

Se eligió un color blue light #03A9F4 como color primario, a partir de éste se obtiene el color primario oscuro. Todos los colores básicos (blue light lo es) de Material Design están ponderados con el número 500, para elegir el primario oscuro debemos de tomar un tono 200 puntos por encima en la paleta de la tonalidad, es decir el número 700 de la paleta de azules.

Una vez obtenidos los dos colores primarios se debe de elegir el color de acentuación, en nuestro caso se creyó conveniente que el “Amber” podía quedar bien y mostrar al usuario de forma inequívoca cuando está seleccionado y a la vez resaltar los detalles.

Respecto al patrón de diseño, se ha empleado el patrón maestro detalle para el menú de navegación.

La ubicación de la lista de elementos es idéntica en cada plataforma; al seleccionar uno de los elementos, se le lleva a la página de detalles correspondiente. Además, la página maestra también incluye una barra de navegación que contiene un botón que se puede usar para ir a la página de detalles activa:

En iOS, la barra de navegación se encuentra en la parte superior de la página y tiene un botón que lleva a la página de detalles. Además, se puede ir a la página de detalles activa si se desliza la página maestra hacia la izquierda.

En Android, la barra de navegación se encuentra en la parte superior de la página y presenta un título, un icono y un botón que lleva a la página de detalles. El icono se define en el atributo [Activity] que decora la clase MainActivity en el proyecto específico de la plataforma Android. Además, se puede ir a la página de detalles activa si se desliza la página maestra hacia la izquierda, si se puntea en el extremo derecho de la pantalla en la página de detalles y si se pulsa el botón Atrás situado en la parte inferior de la pantalla.

En Plataforma universal de Windows (UWP), la barra de navegación se encuentra en la parte superior de la página y tiene un botón que lleva a la página de detalles.

Una página de detalles presenta datos correspondientes al elemento seleccionado en la página maestra; los componentes principales de la página de detalles se muestran en las capturas de pantalla siguientes:

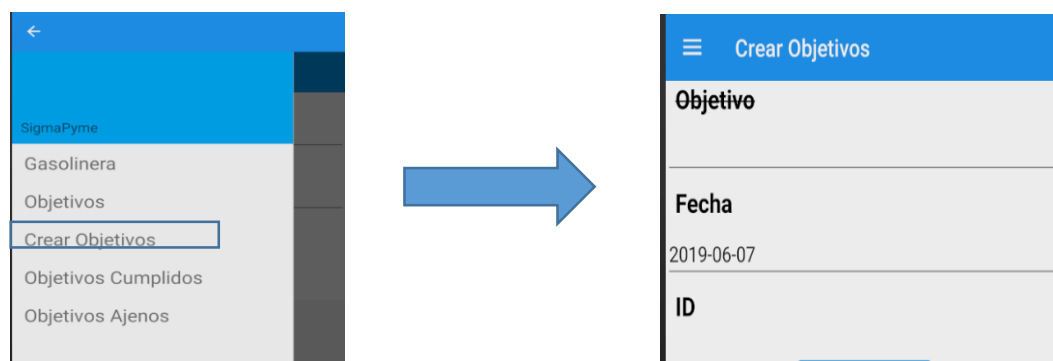
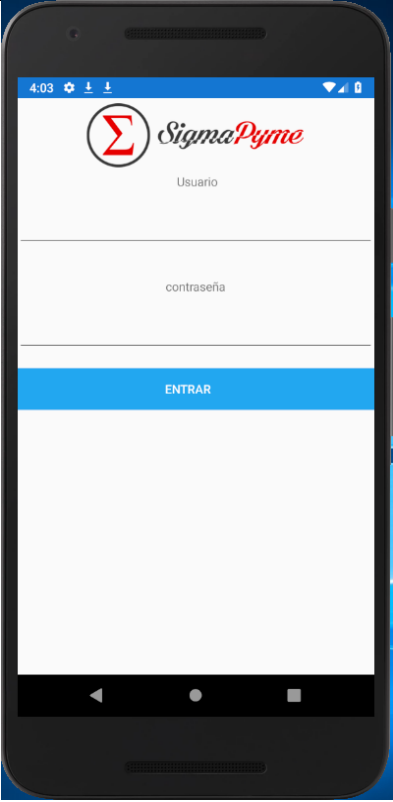


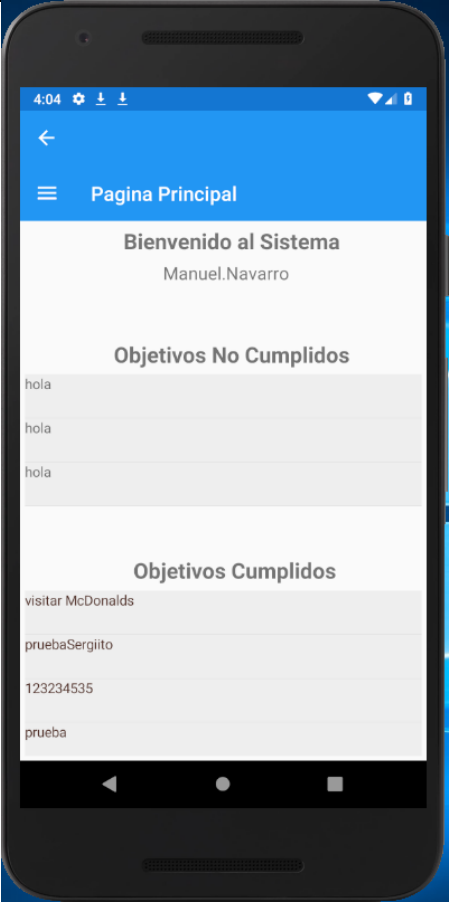
Ilustración 22 Explicacion Diseño App

A continuación, vamos a realizar una explicación detallada de las pantallas que componen la Aplicación:

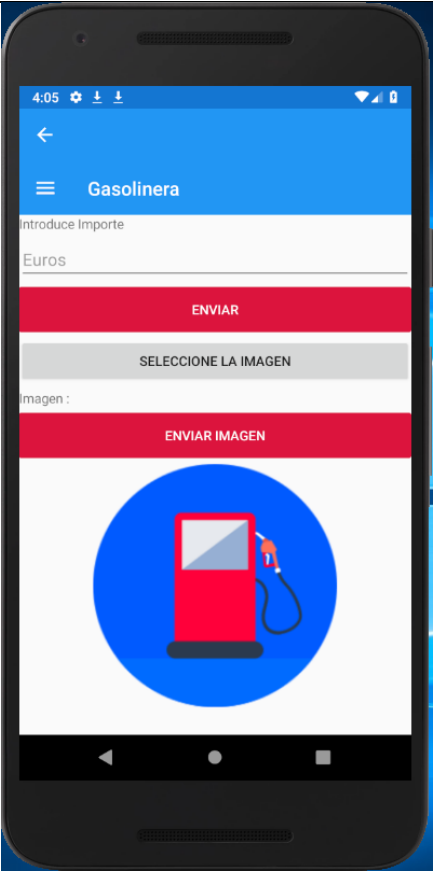
Inicio de Sesión

Descripción	Es la pantalla inicial de la aplicación cuando no estemos autenticados, esta pantalla nos permitirá acceder a la aplicación mediante un logueo.
Activación	Al abrir la aplicación nos saldrá esta pantalla.
Diseño	 <p><i>Ilustración 23 Diseño Inicio Sesión</i></p>
Eventos	<ul style="list-style-type: none"> • Rellenar Formulario para poder Iniciar sesión • Pulsar sobre Login para Iniciar Sesión

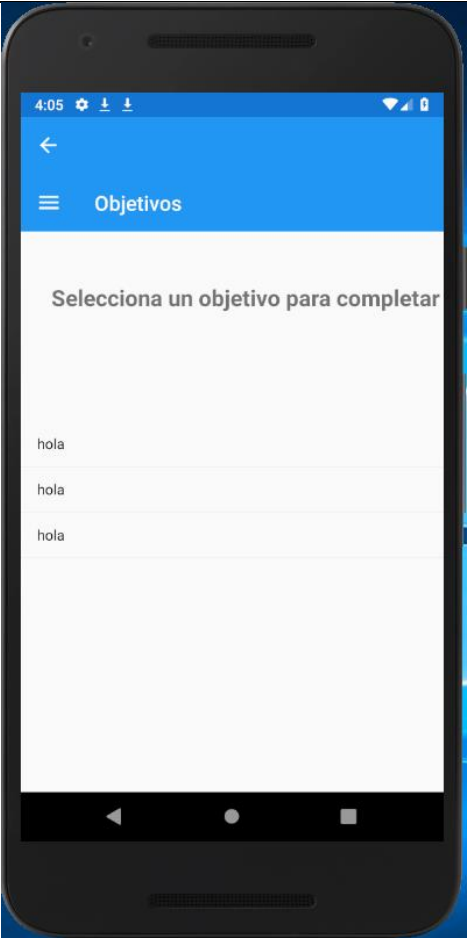
Página Principal

Descripción	Esta es la Página Principal donde se mostrara quien es el que ha iniciado sesión.
Activación	Al acceder a la aplicación con las credenciales correctas nos saldrá esta pantalla.
Diseño	
Eventos	

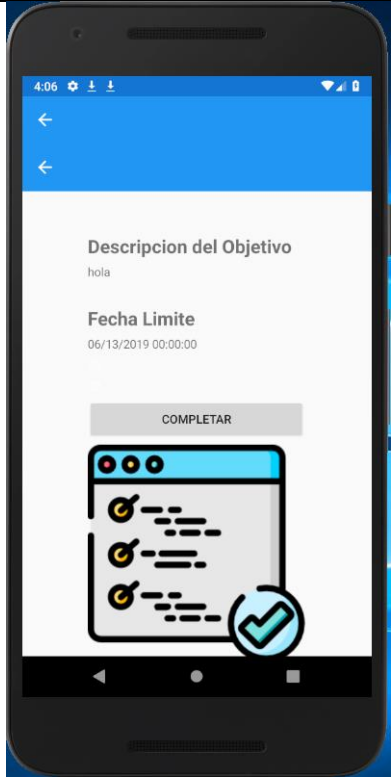
Gasolinera

Descripción	Esta es la página del formulario de la gasolinera donde el usuario introducirá el importe de gasolina que ha echado y seleccionará una imagen para enviar su ticket por correo.
Activación	En el menú seleccionaremos Gasolinera para mostrar esta pantalla
Diseño	 <p><i>Ilustración 25 Diseño Gasolinera</i></p>
Eventos	<ul style="list-style-type: none"> • Introducir Importe • Seleccionar Imagen • Pulsar botón enviar

Objetivos

Descripción	Esta es la página de objetivos , el usuario podrá ver todos los objetivos que tiene pendientes y clickar en ellos para acceder a una página de detalles del objetivo
Activación	En el menú seleccionaremos Objetivos para mostrar esta pantalla
Diseño	 <p><i>Ilustración 26 Diseño Objetivos</i></p>
Eventos	<ul style="list-style-type: none"> Clickar en el objetivo que queramos ver o completar

Detalle Objetivos

Descripción	Esta es la página de detalles del objetivo donde podremos ver los datos del objetivo y darle a completar
Activación	En la ventana objetivo seleccionaremos el objetivo que queramos ver.
Diseño	 <p><i>Ilustración 27 Diseño Detalles Objetivos</i></p>
Eventos	<ul style="list-style-type: none"> Haremos clic en completar para completar el objetivo.

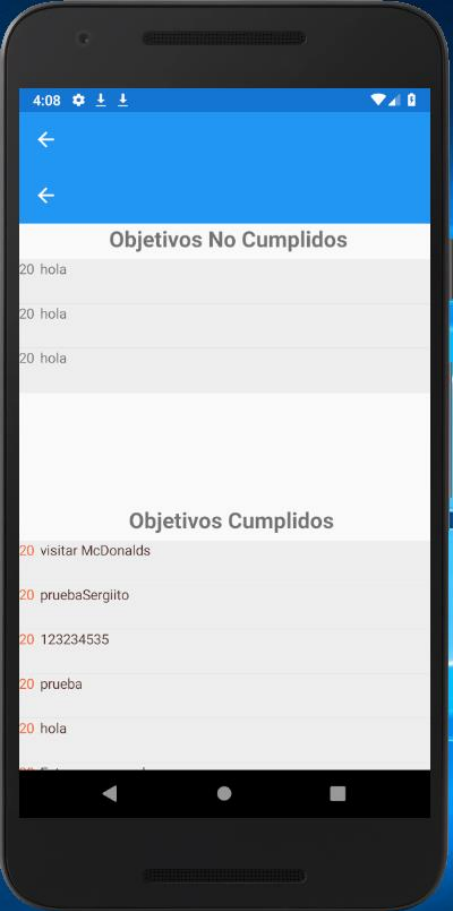
Crear Objetivos

Descripción	Esta función solo está permitida a los administradores, solo ellos podrán crear objetivos.
Activación	En el menú seleccionaremos Crear objetivos para acceder a esta ventana.
Diseño	 <p><i>Ilustración 28 Diseño crear Objetivos</i></p>
Eventos	<ul style="list-style-type: none"> • Rellenamos el formulario • Escogeremos a que persona enviaremos el objetivo clicando en ella. • Pulsaremos en el botón crear Objetivo.

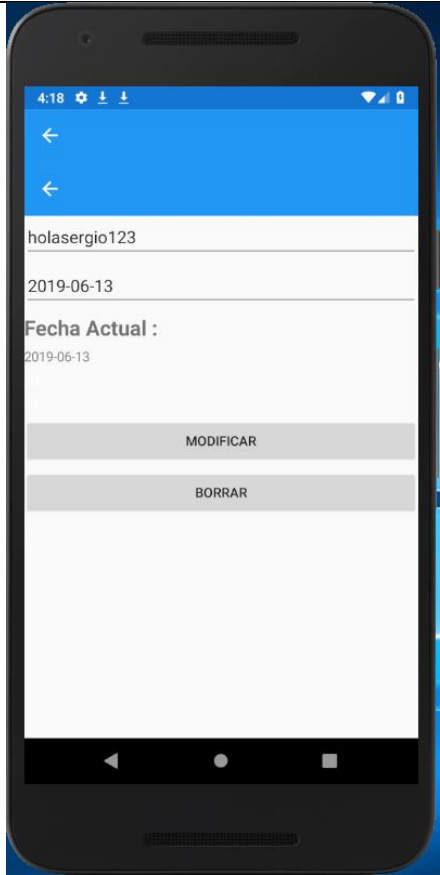
Objetivos Ajenos

Descripción	Esta ventana es solo para Administradores en ella el Administrador podrá ver los objetivos de cualquier usuario para ver que lleva realizado y que no.
Activación	En el menú seleccionaremos Objetivos Ajenos para acceder a esta ventana.
Diseño	 <p><i>Ilustración 30 Diseño Objetivos Ajenos</i></p>
Eventos	<ul style="list-style-type: none"> • Seleccionar el usuario al que queremos verle los objetivos

Objetivos Ajenos	Descripción
------------------	-------------

Descripción	Esta ventana es solo para Administradores en ella el Administrador podrá ver los objetivos de cualquier usuario para ver que lleva realizado y que no.
Activación	Seleccionar el usuario al que queremos verle los objetivos.
Diseño	 <p><i>Ilustración 31 Diseño Objetivos Ajenos Descripción</i></p>
Eventos	<ul style="list-style-type: none"> Vista de objetivos cumplidos y no cumplidos del usuario elegido.

Detalles Objetivos Ajenos

Descripción	Esta ventana permitirá al administrador poder borrar o modificar algún objetivo ya creado
Activación	Clicaremos en uno de los objetivos mostrados en la página mostrar objetivos ajenos
Diseño	
Eventos	<ul style="list-style-type: none"> • Permite modificar un objetivo • Permite borrar un objetivo

IMPLEMENTACIÓN

4.1 Herramientas empleadas

Visual Studio: Es un editor de código ligero desarrollado por Microsoft, inicialmente en exclusiva para Windows pero que actualmente cuenta con soporte multiplataforma. Tiene embebido soporte para control de versiones

StarUML: Se trata de una herramienta que permite realizar de diagramas UML de todo tipo.

Github: Es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el control de versiones Git.

JSON: JavaScript Object Notation, es un formato de texto ligero estandarizado a menudo empleado cuando hay servicios RES entre otros de por medio.

XML: Extensible Market Language, se trata del lenguaje utilizado para la definición de todas las vistas de la aplicación.

4.2 Requisitos Hardware y Software

Estos son los requisitos mínimos que deberá tener un Smartphone o Tablet para poder correr correctamente la aplicación.

Requisitos Software	Sistema Operativo	1.2GHz Dual-Core
	Otros	1.5Gb
Requisitos Hardware	CPU	Conexión a Internet
	RAM	Android Api 23 o superior
	Otros	-

Ilustración 32 Requisitos Hardware y Software

Codificación de las diferentes capas

Para este proyecto he utilizado el patrón MVVM normalmente utilizado en las aplicaciones de Xamarin.

La finalidad principal del patrón MVVM (Modelo Vista Vista-Modelo) es tratar de desacoplar lo máximo posible la interfaz de usuario de la lógica de la aplicación. Veamos a grandes rasgos sus partes principales:

4.2.1 El modelo

Representa la capa de datos y/o la lógica de negocio, también denominado como el objeto del dominio. El modelo contiene la información, pero nunca las acciones o servicios que la manipulan. En ningún caso tiene dependencia alguna con la vista.

Ejemplo:

```

using System;
using System.Collections.Generic;
using System.Text;

namespace SergioDelgadoProyecto.ServicioRest
{
    public class Model_Post
    {
        public int id { get; set; }
        public string objetivos { get; set; }
        public DateTime fecha { get; set; }
        public int registro_id { get; set; }
    }
}

```

Ilustración 33 Ejemplo Model

4.2.2 La vista

La misión de la vista es representar la información a través de los elementos visuales que la componen. Las vistas en MVVM son activas, contienen comportamientos, eventos y enlaces a datos que, en cierta manera, necesitan tener conocimiento del modelo subyacente. En Xamarin Forms podemos crear nuestras interfaces a través de código C# o XAML.

Ejemplo

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="SergioDelgadoProyecto.formGasolinera.Gasolinera">
    <ContentPage.Content>
        <StackLayout>
            <StackLayout>
                <Grid>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*" />
                    </Grid.ColumnDefinitions>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="*" />
                    </Grid.RowDefinitions>
                </Grid>
                <Label Text="Introduce Importe" />
                <Entry x:Name="envioImporte" Placeholder="Euros" />
                <Button Text="Seleccione la Imagen" x:Name="fileButton" Clicked="File"/>
                <Label Text="Imagen : " x:Name="fileLabel"/>
                <Button Text="Enviar" Clicked="SendEmailAsync" BackgroundColor="Crimson" TextColor="White" WidthRequest=
            </StackLayout>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>

```

Ilustración 34 Ejemplo View

4.2.3 Modelo de vista (ViewModel)

El ViewModel (modelo de vista) es un actor intermediario entre el modelo y la vista, contiene toda la lógica de presentación y se comporta como una abstracción de la interfaz. La comunicación entre la vista y el viewmodel se realiza por medio los enlaces de datos (binders).

Ejemplo:

```
public class RestServiceRegistro : IRestServiceRegistro
{
    HttpClient _client;

    public List<registros> Items { get; private set; }

    public RestServiceRegistro()
    {
        _client = new HttpClient();
    }

    public async Task<List<registros>> RefreshDataAsync()
    {
        Items = new List<registros>();

        var uri = new Uri(string.Format("", string.Empty));
        try
        {
            var response = await _client.GetAsync(uri);
            if (response.IsSuccessStatusCode)
            {
                var content = await response.Content.ReadAsStringAsync();
                Items = JsonConvert.DeserializeObject<List<registros>>(content);
            }
        }
    }
}
```

Ilustración 35 Ejemplo viewModel



Ilustración 36 Imagen MVVM

4.3 Acceso a Base de datos

Para acceder a nuestra base de datos he utilizado un servicio Rest:

REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad, pero también mucha complejidad. A veces es preferible una solución más sencilla de manipulación de datos como REST.

Este servicio lo hemos dividido en cuatro carpetas:





 proyectobdSigma	18/05/2019 18:27	Carpeta de archivos
 sigmaP	28/03/2019 9:27	Carpeta de archivos
 sigmaPCumplidos	02/06/2019 13:15	Carpeta de archivos
 sigmaPUusuarios	01/06/2019 21:00	Carpeta de archivos

Ilustración 37 Carpetas Servicio Rest

Cada una de ellas está destinada a una parte de la aplicación:

ProyectobdSigma

Descripción	<p>Esta parte del servicio se encarga de la visualización y creación de los objetivos</p> <p>Contará con:</p> <ul style="list-style-type: none">• Db.php: Conexión a la base de datos.• Modelo: Es donde se encontrarán los diferentes campos de la base de datos.• DBHelper: Es donde estarán las sentencias SQL para acceder a la base de datos insertar, modificar, borrar y mostrar.• Index.php: Es donde se encontrará las operaciones de nuestro sistema REST
Modelo	A continuación, vamos a mostrar una visualización de nuestro modelo:

	<pre> public \$objetivos; public \$fecha; public \$cumplido; public \$registro_id; public function __construct(\$objetivos='', \$fecha='', \$cumplido='', \$registro_id='') { \$this->objetivos = \$objetivos; \$this->fecha = \$fecha; \$this->cumplido = \$cumplido; \$this->registro_id = \$registro_id; } </pre>
DBHelper	<p>A continuación, vamos a mostrar una visualización de nuestro DBHelper :</p> <pre> /** * Devuelve todos los elementos almacenados */ public function getAll() { \$sql = "select * from objetivos order by id"; \$this->logDebug("DB_Prov::getAll \n" . \$sql); \$rs = \$this->db->query(\$sql); return \$rs->fetchAll(PDO::FETCH_OBJ); } /** * Devuelve el elemento con indice id * @param type \$cod * @return record null */ public function get(\$id) { \$sql = "select * from objetivos where registro_id=\$id and cumplido"; \$this->logDebug("DB_Prov::get \n" . \$sql); \$rs = \$this->db->query(\$sql); return \$rs->fetchAll(PDO::FETCH_OBJ); } /** * Devuelve el indice de un código en la lista * @param type \$cod * @return record null */ public function getByname(\$nombre) { \$sql = "select * from objetivos where objetivos='\$nombre'"; \$this->logDebug("DB_Prov::getByCod \n" . \$sql); \$rs = \$this->db->query(\$sql); return \$rs->fetch(PDO::FETCH_OBJ); } </pre>
Index.php	<p>A continuación, vamos a mostrar una visualización de nuestro index.php</p>

	<pre> \$app->get('/objetivos', function (Request \$request, Response \$response, array \$args) use(\$db_prov) { \$this["logger"]->debug('GET /objetivos'); try { return \$response->withJson(\$db_prov->getAll()); } catch (Exception \$e) { \$this["logger"]->error("Error: {\$e->getMessage()}"); return \$response->withJson(['error' => 1, 'desc' => 'Error procesando petición' . \$e->getMessage(), 400); // Véase códigos de error https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP } }); </pre>
--	--

sigmaPCumplidos	
Descripción	<p>Esta parte del servicio se encarga de la visualización de objetivos ya cumplidos, es decir aquellos que en la base de datos en el campo cumplido tienen como valor 1.</p> <p>Contará con:</p> <ul style="list-style-type: none"> • Db.php: Conexión a la base de datos. • Modelo: Es donde se encontrarán los diferentes campos de la base de datos. • DBHelper: Es donde estarán las sentencias SQL para acceder a la base de datos insertar, modificar, borrar y mostrar. • Index.php: Es donde se encontrará las operaciones de nuestro sistema REST
Modelo	<p>A continuación, vamos a mostrar una visualización de nuestro modelo:</p> <pre> public \$objetivos; public \$fecha; public \$cumplido; public \$registro_id; public function __construct(\$objetivos='', \$fecha='', \$cumplido='', \$registro_ \$this->objetivos = \$objetivos; \$this->fecha = \$fecha; \$this->cumplido = \$cumplido; \$this->registro_id = \$registro_id; } </pre>
DBHelper	<p>A continuación, vamos a mostrar una visualización de nuestro DBHelper :</p>

	<pre> public function get(\$id) { \$sql = "select * from objetivos where registro_id=\$id and cumplido='1'"; \$this->logDebug("DB_Prov::get \n" . \$sql); \$rs = \$this->db->query(\$sql); return \$rs->fetchAll(PDO::FETCH_OBJ); } /** * Devuelve el índice de un código en la lista * @param type \$cod * @return record null */ public function getByname(\$nombre) { \$sql = "select * from objetivos where objetivos='\$nombre'"; \$this->logDebug("DB_Prov::getByCod \n" . \$sql); \$rs = \$this->db->query(\$sql); return \$rs->fetch(PDO::FETCH_OBJ); } </pre>
Index.php	<p>A continuación, vamos a mostrar una visualización de nuestro index.php</p> <pre> \$app->get('/objetivo/{id}', function (Request \$request, Response \$response, array \$args) use(\$db_prov) { \$id = \$args['id']; \$this["logger"]->debug('GET /objetivo/' . \$id); try { \$objetivo = \$db_prov->get(\$id); } catch (Exception \$e) { \$this["logger"]->error("Error: {\$e->getMessage()}"); return \$response->withJson(['error'=>1, 'desc' => 'Error procesando petición' . \$e->getMessage()], 400); // Véase códigos de error https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP } if (\$objetivo) { // Existe la provincia devolvamos su valor return \$response->withJson(\$db_prov->get(\$id)); } else { // No existe el elemento, devolvemos código 404 return \$response->withJson(\$db_prov->get(40)); } }); </pre>

sigmaPUusuarios

Descripción	<p>Esta parte del servicio se encarga de la visualización de objetivos ya cumplidos, es decir aquellos que en la base de datos en el campo cumplido tienen como valor 1.</p> <p>Contará con:</p> <ul style="list-style-type: none"> • Db.php: Conexión a la base de datos. • Modelo: Es donde se encontrarán los diferentes campos de la base de datos. • DBHelper: Es donde estarán las sentencias SQL para acceder a la base de datos insertar, modificar, borrar y mostrar. • Index.php: Es donde se encontrará las operaciones de nuestro sistema REST

sigmaP

Descripción	<p>En esta carpeta encontraremos login.php que será por el cual accederemos a nuestra aplicación.</p> <p>Contará con:</p> <p>Login.php : archivo en el cual está el acceso para iniciar sesión en nuestra aplicación .</p>
Login.php	<pre> <?php \$conn = new mysqli('localhost', 'root', ''); mysqli_select_db(\$conn, 'sigmapyme'); if (isset(\$_GET['username']) && \$_GET['username'] != '' &&isset(\$_GET['password']) && \$_GET['password'] != '') { \$name = \$_GET['username']; \$password = sha1(\$_GET['password']); \$registro="registro"; \$getData = "SELECT id,nombre,rol FROM \$registro WHERE nombre='\$name' and contrasena='\$password'"; \$result = mysqli_query(\$conn,\$getData); if(\$result === false) { echo "error while executing mysql: " . mysqli_error(\$conn); } while(\$r = mysqli_fetch_row(\$result)) { \$userId=\$r[0]; \$userName=\$r[1]; \$userRol=\$r[2]; } if (\$result->num_rows> 0){ \$resp["id"]=\$userId; \$resp["usuario"]=\$userName; \$resp["contrasena"]="Secreto"; \$resp["rol"]=\$userRol; // \$resp[\$userRol]= \$userRol; // \$resp["message"] = "Login successfully"; } else { \$resp["id"]= 0; \$resp["usuario"]="error"; \$resp["contrasena"]="error"; \$resp["rol"]="error"; } } </pre>

Estos serían los archivos que forman nuestro servicio REST que se conectarán con nuestra base de datos para realizar las sentencias determinadas.

4.4 Estructura del proyecto

En esta sección se detalla cómo es la estructura interna del proyecto, es decir, cómo están organizados los directorios de archivos que componen el proyecto.

El código fuente de la aplicación se encuentra se encuentra en el directorio del proyecto:

`\SergioDelgadoProyecto\SergioDelgadoProyecto\SergioDelgadoProyecto\`

Las dependencias de nuestro proyecto se encuentran en la carpeta con ese mismo nombre

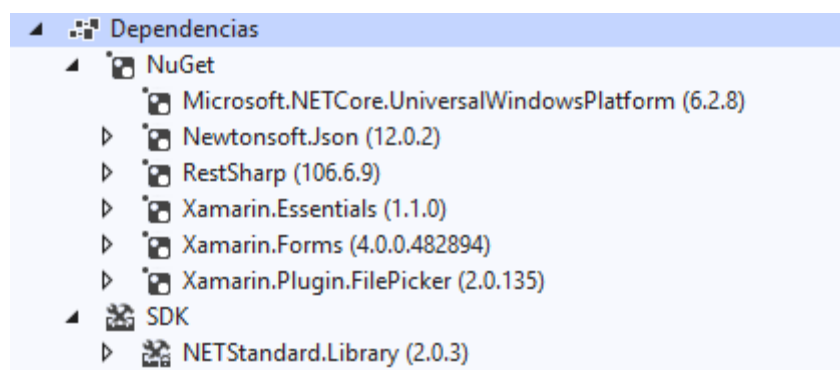


Ilustración 38 Dependencias proyecto

Aquí vemos todas las dependencias que hemos utilizado en nuestro proyecto:

Newtonsoft.Json: Es la utiliza para poder leer los archivos Json que provienen de nuestro servicio Rest

RestSharp: esta librería se utiliza para facilitarnos el control del servicio Rest

Xamarin.Essentials: esta librería es necesaria para poder trabajar con Xamarin , contiene todo lo necesario para ello.

Xamarin.Forms: esta librería es para trabajar con los forms de Xamarin

Xamarin.Plugin.FilePicker: esta librería es la que utilizamos para poder coger datos del almacenamiento de nuestro teléfono.

En la carpeta Servicio Rest encontraremos nuestros Modelos y nuestros Vista-Modelos

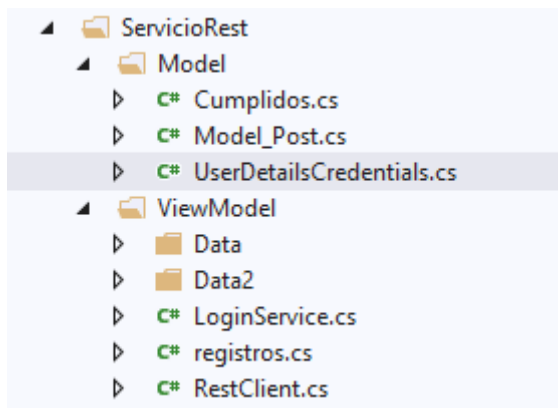


Ilustración 39 Carpeta Servicio Rest proyecto

Model:

En este directorio encontramos las clases relativas a la definición de los modelos entidad utilizados en la aplicación y cuya definición es estándar para que Mysql los procese internamente como objetos en la base de datos, en nuestro caso la definición Cumplidos, Model_post, UserDetailsCredentials y registros.

Cumplidos: Clase que contiene todos los registros de la tabla Objetivos

```
namespace SergioDelgadoProyecto.ServicioRest
{
    class Cumplidos
    {
        public int id { get; set; }
        public string objetivos { get; set; }
        public DateTime fecha { get; set; }
        public int cumplido { get; set; }
        public int registro_id { get; set; }
    }
}
```

Model_Post: Clase que contiene todos los registros de la tabla Objetivo exceptuando el registro cumplido

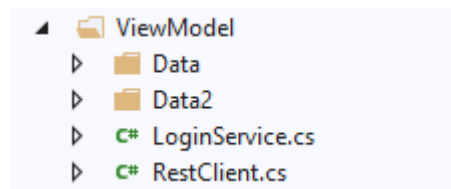
```
namespace SergioDelgadoProyecto.ServicioRest
{
    public class Model_Post
    {
        public int id { get; set; }
        public string objetivos { get; set; }
        public DateTime fecha { get; set; }
        public int registro_id { get; set; }
    }
}
```

UserDetailsCredentials: clase que contiene todos los registros de la tabla registros

```
namespace SergioDelgadoProyecto
{
    public class UserDetailsCredentials
    {
        [JsonProperty("id")]
        public int id { get; set; }
        [JsonProperty("usuario")]
        public string usuario { get; set; }
        [JsonProperty("contrasena")]
        public string contrasena { get; set; }
        [JsonProperty("rol")]
        public string rol { get; set; }
    }
}
```

Estos modelos son necesarios para poder recoger o enviar los datos que vienen de la base de datos.

Luego tendremos la carpeta de ViewModel



Donde se encontrará todo lo relacionado al servicio rest de la aplicación y el login.

\SergioDelgadoProyecto\view

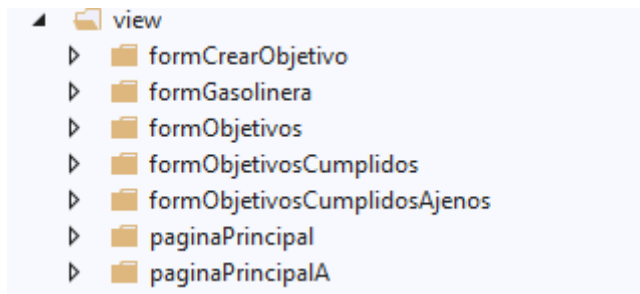


Ilustración 40 carpetas de vistas proyecto

En este directorio se organizan todas clases relativas a la parte lógica de la aplicación y contenedoras de la interfaz gráfica.

- formCrearObjetivo
- formGasolinera
- formObjetivosCumplidos
- formObjetivosCumplidosAjenos
- paginaPrincipal
- paginaPrincipalA

Pruebas

5.1 Prueba de caja blanca

Las pruebas de caja negra consisten en la prueba individual de cada uno de las funcionalidades internas. Son principalmente pruebas unitarias y de integración, en ellas se comprueban todos los posibles caminos lógicos que pueden tomar los métodos o los diferentes tipos de valores que se pueden pasar.

Todas estas pruebas se realizan durante el tiempo de implementación y no a posteriori cómo las de caja negra.

Las pruebas de caja blanca que se han realizado a lo largo de este proyecto se resumen en:

- Comprobación de la validación de entrada de datos en cada uno de los formularios.
- Comprobación de las salidas del sistema tras la introducción forzosa de datos truncados en la base de datos.
- Comprobación de la comunicación del sistema en los diferentes escenarios posibles.
- Comprobación del funcionamiento de los servicios de manera aislada.
- Comprobación del control de la sesión.
- Comprobación de la concordancia entre las entradas de datos y la información guardada en Mysql.
- Comprobación del comportamiento de la caché de la aplicación.

5.2 Pruebas de caja negra

Las pruebas de caja negra son un método de prueba de software en el que la estructura/diseño/implementación interna del ítem que se está probando no es conocida por el probador. Estas pruebas pueden ser funcionales o no funcionales, aunque generalmente funcionales. Esa es la diferencia principal respecto a las pruebas de caja blanca.

Por ello, para la realización de estas pruebas se ha solicitado a compañeros de trabajo cuyo conocimiento del proyecto era nulo, que sean ellos quien prueben el sistema limitando mi participación a la anotación de los resultados. Todas estas pruebas han sido realizadas obviamente después de haber sido implementadas las funcionalidades en el sistema.

Veamos una descripción de las pruebas que se han realizado:

Inicio de sesión

Objetivo	Realizar el inicio de Sesión
Entrada de datos	Usuario: Manuel.Navarro Contraseña: Manna@2019
Postcondiciones esperadas	<ul style="list-style-type: none"> • Completar el Inicio de sesión • Ir a la pantalla principal de la aplicación
Secuencia	<ul style="list-style-type: none"> • Rellenar formulario • Realizar validaciones • Enviar Información al servicio • Recibir respuesta • Realizar acciones consecuentes
Resultado	El resultado fue satisfactorio

Ilustración 41 tabla pruebas - Inicio Sesión

Truncar Inicio de sesión

Objetivo	Comprobar el funcionamiento de las validaciones del sistema cuando se realiza el inicio de sesión
Precondiciones	No tener una sesión activa
Entrada de datos	Usuario: Manuel.Navarro Contraseña: Manna@2019asfafsfa
Postcondiciones esperadas	<ul style="list-style-type: none"> • Las validaciones que se realizan en el cliente impide el envío de datos al servicio • Se muestra aviso visual
Secuencia	<ul style="list-style-type: none"> • Rellenar formulario • Realizar validaciones • Mostrar aviso visual del resultado
Resultado	El resultado fue satisfactorio

Ilustración 42 tabla pruebas - Truncar Inicio Sesión

Visualizar Objetivos

Objetivo	Comprobar el funcionamiento del listview que contiene la lista de objetivos
Precondiciones	Tener una sesión activa y encontrarse en el fragmento “Objetivos”.
Entrada de datos	Ninguna
Secuencia	<ul style="list-style-type: none"> • Acceder a la aplicación con las credenciales necesarias • Ir a la pestaña objetivo
Resultado	El resultado fue satisfactorio

Ilustración 43 tabla pruebas - Visualizar Objetivos

Crear Objetivos

Objetivo	Comprobar el funcionamiento de crear objetivos
Precondiciones	Tener una sesión activa y estar autenticado como administrador , además de encontrarse en el fragmento “crear objetivos”.
Entrada de datos	Ninguna
Secuencia	<ul style="list-style-type: none"> • Acceder a la aplicación con las credenciales necesarias • Ir a la pestaña crear Objetivo • Rellenar formulario
Resultado	El resultado fue satisfactorio

Ilustración 44 tabla pruebas - crear objetivos

Formulario Gasolinera

Objetivo	Comprobar el funcionamiento del formulario de Gasolinera.
Precondiciones	Tener una sesión activa y encontrarse en el fragmento “Gasolinera”.
Entrada de datos	Ninguna
Secuencia	<ul style="list-style-type: none"> • Acceder a la aplicación con las credenciales necesarias • Ir a la pestaña crear Objetivo • Rellenar formulario
Resultado	El resultado fue satisfactorio

Ilustración 45 tabla pruebas - formulario gasolinera

Visualizar Objetivos Ajenos

Objetivo	Comprobar el funcionamiento del listview que contiene la lista de objetivos Ajenos
Precondiciones	Tener una sesión activa y encontrarse en el fragmento “objetivos Ajenos”.
Entrada de datos	Ninguna
Secuencia	<ul style="list-style-type: none"> • Acceder a la aplicación con las credenciales necesarias y siendo administrador • Ir a la pestaña objetivos Ajenos • Pulsar en el nombre de usuario al que quieras verle los objetivos
Resultado	El resultado fue satisfactorio

Ilustración 46 tabla pruebas - visualizar objetivos ajenos

Visualizar Objetivos Completados

Objetivo	Comprobar el funcionamiento del listview que contiene la lista de objetivos completados
Precondiciones	Tener una sesión activa y encontrarse en el fragmento “objetivos completados”.
Entrada de datos	Ninguna
Secuencia	<ul style="list-style-type: none"> • Acceder a la aplicación con las credenciales necesarias • Ir a la pestaña objetivos Ajenos
Resultado	El resultado fue satisfactorio

Ilustración 47 tabla pruebas visualizar objetivos completados

Conclusiones

6.1 Conclusión

A lo largo de la implementación del sistema se han encontrado numerosos inconvenientes que en su mayoría terminaron resolviéndose con horas de investigación, estudio o simplemente prueba y error.

En su mayoría, estas dificultades vinieron dadas en relación a xamarin dado que es un lenguaje el cual aún tiene muy poca información y sobre todo muy poco apoyo por las demás plataformas (Google, azure ...). Mi intención en un primer momento era haber usado Firebase pero al no tener apoyo de Google decidí coger Mysql.

A pesar de todo lo anterior, puedo concluir que ha sido un proyecto realmente satisfactorio, tanto a nivel personal, por el duro camino. Han sido muchos días de obtener información para resolver los problemas que me iba encontrando y sobre todo ver el alcance real de un proyecto.

También a nivel académico, he podido entender mejor como funciona visual Studio y he aprendido un poco de Xamarin. Con ello he afianzado los conocimientos en programación en C# que tenía y ha creado la inquietud de seguir aprendiendo sobre esta tecnología.

6.2 Mejoras Futuras

Si bien es cierto que cuanto más avanzado estaba el proyecto y más conocimiento había sido adquirido, el abanico de opciones se multiplicaba y muchas grandes ideas tuvieron que ser desechadas, por lo avanzado del mismo y por la falta del tiempo que habría sido necesario para llevarlas a cabo. Algunas de ellas son las siguientes:

- Notificaciones Push.
- Mejor diseño de la aplicación.
- Añadir más ventanas.
- Añadir un control horario.
- Mejorar la aplicación.

