UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Desarrollo de una aplicación multiplataforma basada en ERP Prologic.

Autor/es

AYOUB BOUSTTA

Director/es

ANA ROMERO IBÁÑEZ

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2018-19

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Desarrollo de una aplicación multiplataforma basada en ERP Prologic.

Realizado por:

Ayoub Boustta

Tutelado por:

Ana Romero Ibáñez

Logroño, junio de 2019

Índice

RESUMEN

Hoy en día es complicado encontrar una empresa en España que no cuente con un sistema de planificación de recursos empresariales, conocidos como ERP, ya que facilitan mucho la administración y gestión de una empresa.

Por otro lado, nos encontramos en un momento donde los dispositivos móviles forman totalmente parte de nuestra sociedad, de los cuales la mayoría tienen buenas prestaciones, y llegando incluso a ser una herramienta de trabajo.

Por ello, como objetivo para este trabajo de fin de grado, se estableció unir las dos ideas anteriores: un sistema ERP es un programa bastante extenso y pesado, que siempre necesita un dispositivo con buenas prestaciones para ser ejecutado y, en determinadas situaciones, gran parte del sistema puede ser innecesario. Así que se planteó la idea de desarrollar una aplicación móvil que pueda llevar a cabo rápidas operaciones de gestión, principalmente la gestión de los pedidos de venta, del ERP Prologic, desarrollado y mantenido por la empresa riojana Logical Rioja SL.

El resultado final de este proyecto es una aplicación móvil que se puede ejecutar tanto en Android como en iOS como en dispositivos Windows (ordenadores también) que abarca la gestión de los pedidos de venta y, además, incluye una agenda de usuario donde el cliente del ERP Prologic puede gestionar sus mensajes, notas y tareas.

ABSTRACT

Nowadays, it's difficult to find a company in Spain that doesn't have an enterprise resource planning, known as ERP, because they facilitate the administration and management of a company.

On the other hand, we are in a moment where mobile devices are totally part of our society, and most of them have good performance, becoming a work tool.

Therefore, as objective of this project, it was established to unite the two previous ideas: an ERP system is a rather extensive and heavy program, which always need a device with good performance to be executed and, in certain situations, a large part of the system may be unnecessary. So it was proposed the idea of developing a mobile application that can carry out rapid management operations, mainly the management of sale orders, of the ERP Prologic, developed and maintained by the company Logical Rioja SL from La Rioja.

The final result of this project is a mobile application that can be run on Android, iOS and in Windows devices (computers also) that covers the management of sale orders and, in addition, includes a user schedule where the customer of the ERP Prologic can manage their messages, notes and tasks.

1. INTRODUCCIÓN

1. Antecedentes

Logical Rioja S.L. es una empresa riojana con sede en Logroño dedicada a la consultoría, el desarrollo de soluciones innovadoras en nuevas tecnologías y la fabricación y distribución a nivel nacional e internacional de software parametrizado (proporcionando la implantación y el mantenimiento del mismo). Fundada en el año 1986, ha desarrollado su propio software sectorial (ERP) Prologic para la gestión de la industria de la maquinaria, la industria del mueble y la industria alimentaria y, además, un software de contabilidad empresarial llamado Conta Prologic. Hacia sus clientes, ofrece servicios de soporte y actualizaciones, formación, consultoría, distribución de hardware y servicio técnico personalizado.

Con un gran equipo de más de 30 profesionales, formado por implantadores, técnicos, programadores, administrativos y comerciales ofrece a sus clientes las últimas tecnologías a través de un departamento I+D+I (Investigación + Desarrollo + Innovación).

La empresa, especializada en la industria alimentaria, proporciona a través de Prologic la gestión de las diferentes industrias:

- **Industria alimentaria:**
  - La gestión de compras y ventas.
  - La gestión del proceso de transporte así como la logística.
  - El control del almacén, el proceso de fabricación y el costo del producto.
  - El intercambio electrónico de datos 'EDI' (a través de un desarrollo propio).
  - Sistema de seguimiento y alarmas (acciones a realizar cuando se produce una determinada situación en el proceso de fabricación).
- **Industria de la maquinaria:**
  - La gestión de ventas.
  - Departamentos de venta de maquinaria, recambios, alquiler y de taller.
- **Industria del mueble:**
  - La gestión de compras y ventas.
  - El proceso de fabricación y transporte.
  - El control de almacenes.
  - La gestión del mueble y despieces.

Con varios proyectos entre manos, se ha ido recabando información durante la estancia en prácticas sobre el funcionamiento de Prologic en la industria alimentaria para proporcionar herramientas nuevas a los usuarios, alcanzando la mayor comodidad para ellos. En busca de ello, se consideró la idea de proporcionar acceso a la plataforma desde diferentes puntos o incluso diferentes dispositivos.

Uno de estos proyectos que está en desarrollo consiste en una aplicación web y, por tanto, había una necesidad de tener un acceso básico y limitado a la funcionalidad general del software a través de aplicaciones (Android, iOS y UWP[1]) para poder llevar a cabo operaciones rápidas o de consulta sin necesidad de acceder desde la aplicación de escritorio sobre todo para la gente con movilidad.

2. Justificación del proyecto

De entre todos los profesionales que forman el equipo de Logical Rioja nos encontramos a los comerciales, que son los encargados de llevar a cabo las relaciones exteriores de la empresa, así como los responsables de dar a conocer el producto (en este caso el software Prologic) entre otras cosas.

Son profesionales que están en constante movimiento, trabajando mucho más fuera de la oficina que dentro, y necesitando siempre tener acceso al producto que venden. El software Prologic es un programa de escritorio, que necesita ciertos recursos para ejecutarse, recursos que para un comercial son innecesarios, además de la necesidad de tener un ordenador para tener acceso.

La idea principal es que los comerciales, a la hora de vender el producto o realizar una pequeña prueba de su funcionamiento, no tengan que depender del software al completo, sino que puedan tener una aplicación que pueda usarse en dispositivos móviles (comodidad para los comerciales de trabajar con una Tablet o una PDA[2]) e incluso en un ordenador con pocos recursos que resuma la funcionalidad más importante del software y sirva de precedente para mostrar a los potenciales clientes el funcionamiento.

Adaptamos el mismo argumento que los comerciales para los clientes y los representantes que realizan ventas en el exterior de sus empresas y no necesitan toda la funcionalidad del software (sólo cuando están fuera de la empresa/oficina).

Dejando a un lado todo el trabajo exterior a la empresa, se observan otras razones en el trabajo interior, como las siguientes.

El trabajo con grandes cantidades de cajas/embalajes es algo muy habitual en las empresas, y los clientes de Prologic tienen que ir añadiendo manualmente gran cantidad de información, por ejemplo, de los pedidos, artículos, etc. Un requisito de la aplicación es que sea capaz de leer códigos QR y que sea compatible con NFC. Con esto lo que se busca es que, con la aplicación móvil, se pueda añadir directamente a la aplicación, por ejemplo, la información de una caja

[1] Plataforma Universal de Windows. Hace referencia a una aplicación para el sistema operativo Windows que puede ser lanzada en distintos dispositivos (Tablet, móvil, Xbox) incluido un ordenador, a partir de Windows 8, como una aplicación móvil.

[2] Pequeño dispositivo que combina funciones de ordenador, teléfono/fax, Internet y conexiones de red.

(contenido, kg, emisor o destino, etc.) usando alguna de estas tecnologías (cajas con códigos QR o con chips NFC) brindando a nuestros clientes más facilidades y el ahorro de mucho tiempo.

Además de todo esto, con una aplicación móvil, vamos a tener la oportunidad de tener contacto más directo con nuestros clientes (mediante notificaciones push) comparado con el contacto por correo electrónico, sobre todo para mensajes importantes y acceso a estadísticas, funcionalidad importante para la gerencia de la empresa.

3. Alcance

El objetivo principal del proyecto es desarrollar una aplicación móvil con la funcionalidad más importante y necesaria del software Prologic ERP, principalmente los pedidos, las expediciones y las estadísticas. Con ella, los usuarios podrán seguir realizando su trabajo desde un dispositivo móvil bridándoles de una gran movilidad y comodidad, sobre todo para el trabajo externo.

Es necesario que, tanto la funcionalidad de pedidos como la de expediciones, implique consultar, crear, modificar y eliminar, mientras que la funcionalidad de las estadísticas es sólo de consulta.

En cuanto a la tecnología a utilizar, se va a trabajar con la tecnología .NET usando el lenguaje de programación C# para el desarrollo de la aplicación, el mismo con el que se ha desarrollado el software Prologic ERP, y Visual Basic para el desarrollo de las diferentes funcionalidades en la API, ya que está desarrollada en este lenguaje. Para la consulta y manipulación de información, también se ha usado la misma tecnología que el programa, en este caso, el mismo sistema gestor de bases de datos que el programa, conocido como Microsoft SQL Server.

Es importante destacar que, principalmente, va dirigida a los clientes del programa ERP Prologic (a partir de ahora conocidos también como usuarios, usuarios de la aplicación móvil y operarios, nombre interno definido para realizar las gestiones del cliente conectado a la aplicación), ya que la aplicación va a acceder directamente a la misma información que se accede desde el programa principal, aunque también será accesible por otras personales como los comerciales y el equipo de la propia empresa (Logical Rioja).

4. Metodología

Para llevar a cabo este proyecto, se ha pensado que la mejor manera de hacerlo es usando la metodología ágil de gestión de proyectos Scrum. A pesar de que está basada principalmente en el trabajo en equipo (y en este caso el grupo sólo está formado por una persona), hay varios beneficios que decantan la balanza en favor a esta metodología:

- Los requisitos están poco definidos y seguramente vayan a cambiar.
- El cliente necesita ver resultados rápidamente (para comprobar si es lo esperado).
- Proyecto basado en un software (Prologic) donde se incluyen funcionalidades nuevas y cambios frecuentemente.
- Una vez finalizada la funcionalidad importante, se van a ir agregando otras dependiendo de los plazos.

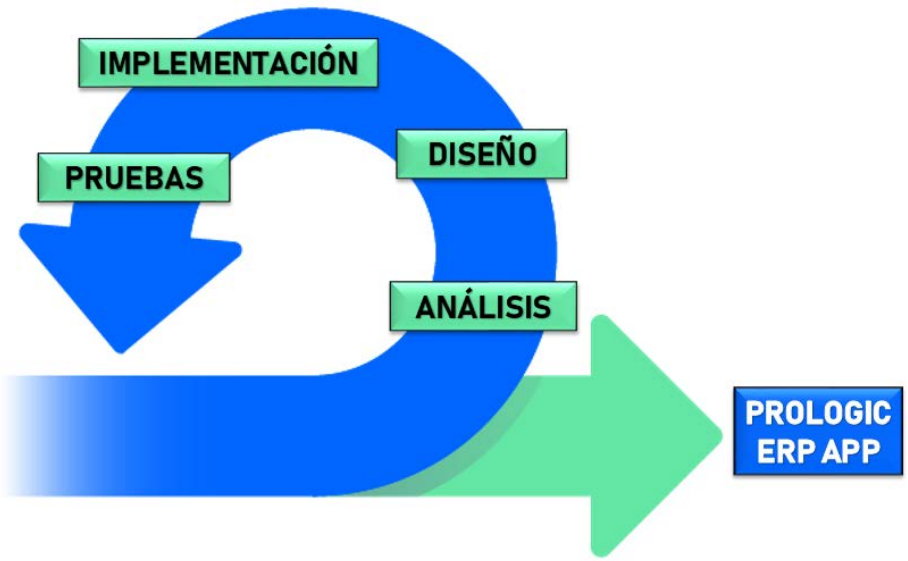

Figura 1. Estructura de las iteraciones del proyecto.

En cada una de las iteraciones establecidas en la tarea de desarrollo, se harán las etapas de análisis, diseño, implementación y pruebas (Figura 1). De esta manera, se podrá adaptar el producto a lo largo de su desarrollo por cambios en los requisitos, causados por la necesidad de integrar otras funcionalidades diferentes o por cambios de mercado, incluso por falta de insatisfacción con el cliente (Ángel Tijero, gerente de Logical Rioja SL), entre otras muchas cosas.

2. FORMACIÓN

Debido a la complejidad tanto del proyecto como del software, ha sido necesario tener una etapa de formación, que comenzó en el plazo de prácticas entrando en contacto con Prologic en la industria alimentaria y entendiendo su lógica.

Una vez finalizado el periodo de prácticas y definido el proyecto a realizar, tuvo lugar un segundo periodo de formación para conocer la tecnología y las herramientas usadas para llevar a cabo la aplicación en las tres plataformas definidas (Android, iOS y UWP).

1. Xamarin

La tecnología usada ha sido .NET de Microsoft usando el lenguaje de programación C# en el IDE[3] Visual Studio, el cual no ha necesitado etapa de formación al tener conocimientos previos.

La herramienta utilizada para el desarrollo del proyecto ha sido Xamarin, integrándola en Visual Studio. La razón principal de la elección de esta herramienta ha sido que permite llevar a cabo el desarrollo multiplataforma a partir de un único lenguaje de programación (C#). Permite reutilizar casi todo el código (90% aproximadamente), ahorrando mucho trabajo y tiempo.

Para iniciar el aprendizaje de Xamarin, se siguieron varios tutoriales que se encuentran en la página oficial de Microsoft (2019) sobre todo para conocer los conceptos básicos y saber cómo se realiza el despliegue de una aplicación en las diferentes plataformas.

En Xamarin se utiliza el patrón de desarrollo Model-View-ViewModel, que es el patrón que se ha utilizado para desarrollar el producto final. Consiste en mantener separada la lógica de negocio de la presentación de la aplicación para su futuro mantenimiento y evolución (sobre todo en aplicaciones como la que se ha desarrollado ya que se espera añadir bastante más funcionalidad una vez que esté distribuida, a partir del feedback de parte de los usuarios).

Existen tres componentes principales en este patrón; el modelo, la vista y el modelo de la vista. La vista, que suele ser dependiente de la plataforma, conoce el modelo de la vista, que es compartida por todas las plataformas, y el modelo de la vista conoce al modelo, que también es compartido por todas las plataformas, pero el modelo no conoce al modelo de vista ni éste a la vista provocando un aislamiento del modelo y la vista. De esta manera, el modelo puede evolucionar independientemente de la vista.

[3] Entorno de desarrollo integrado, programa utilizado para llevar a cabo el desarrollo de software.

Usando de ejemplo la aplicación desarrollada, podríamos establecer el patrón de la siguiente manera:

- La vista serían los archivos de extensión .xaml y .cs que definen como es el diseño de las diferentes ventanas de la aplicación.
- El modelo de la vista, entre otras muchas cosas, realiza el databinding.
- El modelo, entre otras muchas cosas, define los diferentes métodos de acceso a la base de datos para realizar consultas SQL o manipular la información.

La mayor parte de la aplicación se desarrolla en lenguaje C# excepto los archivos .xaml, que se desarrollan en lenguaje XAML[4].

En cuanto al contenido de las pestañas, Xamarin usa diferentes tipos de Layouts para definir la forma en la que se va a mostrar el contenido permitiendo utilizar diferentes a lo largo del desarrollo de la aplicación e incluso utilizar varios tipos anidados en la misma pestaña.

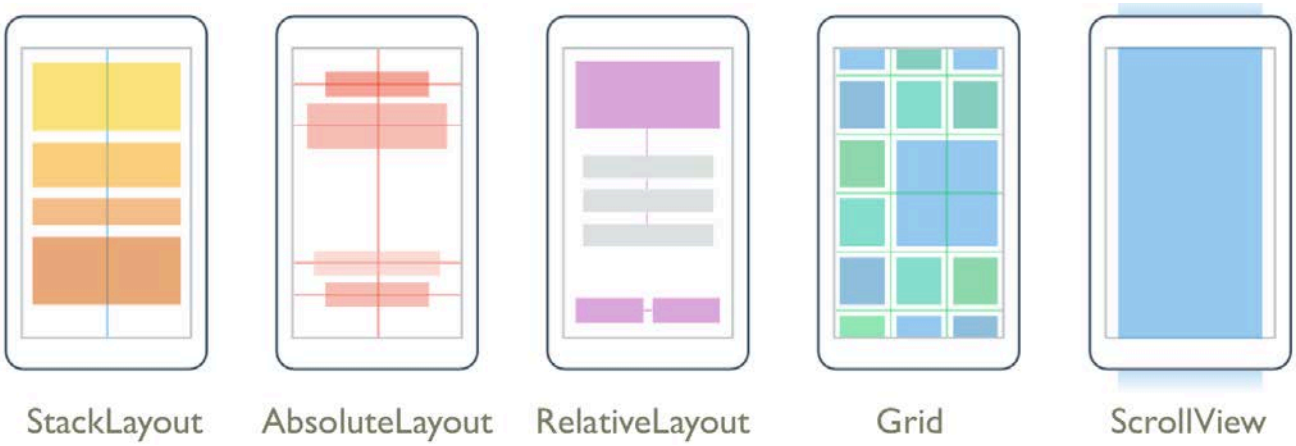

Figura 2. Tipos de Layouts usados en Xamarin.

En primer lugar está el StackLayout, que es el más básico y el más fácil de utilizar. Como bien dice su nombre, organiza los elementos en forma de pila, tanto horizontal como vertical. El AbsoluteLayout coloca los elementos según su tamaño, posición o sus valores absolutos mientras que el RelativeLayout se usa para establecer la posición de los elementos en relación a las propiedades del diseño de la vista o del nivel (son Layouts que siempre están sometidos a comparación, es decir, cuando se habla de uno de ellos también se habla de las diferencias con el otro al tener una lógica bastante parecida). Uno de los más utilizados es el GridLayout, que es como una tabla o una cuadrícula y se organiza en filas y columnas. Por último tenemos al ScrollView, que permite el desplazamiento fuera de la pantalla (es bastante imprescindible a la hora de realizar una aplicación).

[4] Definido como lenguaje de marcado extensible de la aplicación, que es una variante desarrollada por Microsoft.

Es muy común en Xamarin que, en determinados momentos, se desarrollen diferentes códigos dependiendo de la plataforma o del dispositivo donde se ejecute la aplicación, cuya información podemos obtener tanto en XAML con el uso de "onPlatform…" como en C# con "Device.RuntimePlatform" o "Device.Idiom".

También hay propiedades que tienen diferente comportamiento por defecto dependiendo de la plataforma y que podemos cambiar su valor según donde se vaya a ejecutar.

2. Desarrollo del prototipo

Para ir poniendo a prueba los conocimientos que se iban obteniendo, se llevó a cabo el desarrollo de una aplicación basada en los primeros patrones de diseño para utilizarla como base para realizar este proyecto. Como se puede comprobar en esta Figura 3, la aplicación tiene 2 menús; Tabbed y MasterDetail**.**

El Tabbed es un menú de pestañas que, en este caso, ha sido usado para la funcionalidad más importante (pestaña principal, pedidos, expediciones y estadísticas) y que tiene un acceso más directo (aparece en las pestañas principales).

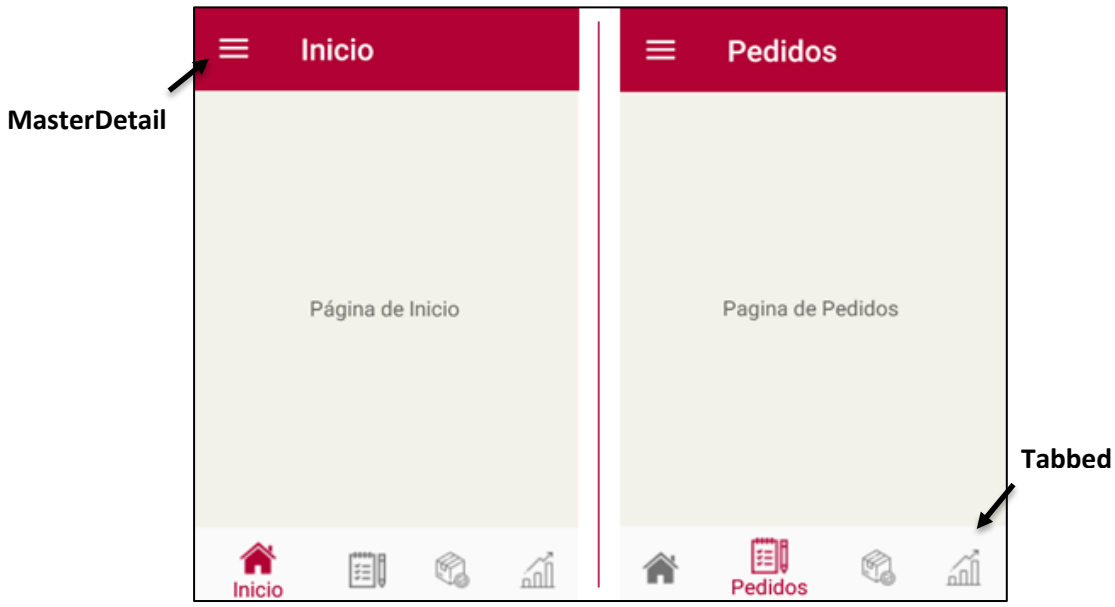

Figura 3. Primera aplicación desarrollada para la formación.

El MasterDetail, conocido coloquialmente por Menú Hamburguesa, es un menú lateral desplegable donde se encuentran otras opciones de navegación o de aplicación (Figura 2) que proporciona, sobre todo, "limpieza" en el diseño ya que no se muestra siempre, sino que se encuentran oculto en el Master.

La página desplegada del menú MasterDetail se denomina Master, y la página principal se denomina Detail. En este caso, el Detail directamente puede ser una página de contenido, o almacenar un tipo de página, como en la Figura 3 que es una página de tipo Tabbed.

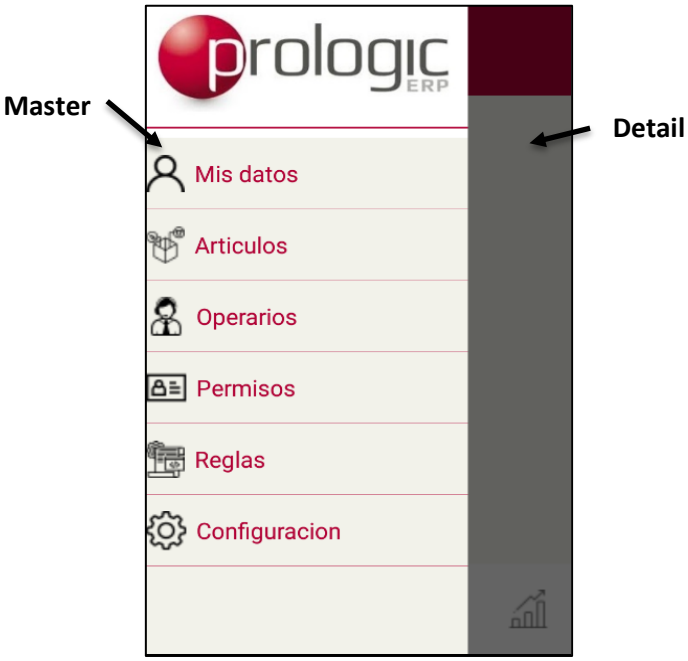

Figura 4. Menú MasterDetail lateral desplegado.

Sobre los Layouts, podemos ver cómo se apilan las diferentes opciones de navegación y el logo de Prologic ERP en la página master de la Figura 4, haciendo uso del StackLayout. En el caso de este Master, se usa para proporcionar otras opciones o funcionalidades de la aplicación que no son las principales.

También se ha hecho uso del GridLayout como se puede comprobar en la Figura 5, al ser uno de los más importantes. En este caso, se ha desarrollado una calculadora con la funcionalidad básica (suma, resta, multiplicación y división) jugando con el relleno de las filas y columnas para realizar la estructura. Este Grid ha proporcionado la capacidad de realizar bastantes cosas en la aplicación, teniendo mayor iniciativa a realizar estructuras, sobre todo anidándolo con otros Layouts.

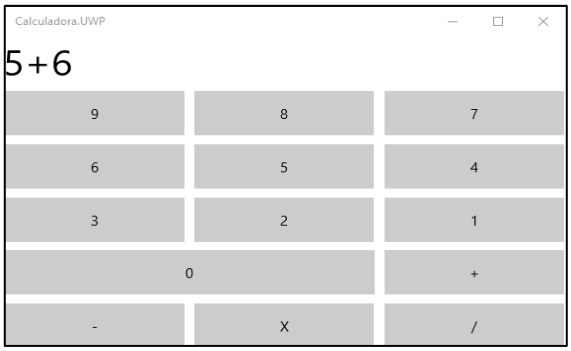

Figura 5. Aplicación UWP Calculadora realizada con un GridLayout para formación.

Esta etapa de formación ha sido necesaria para ir conociendo la herramienta y la forma de trabajar pero cabe destacar que la formación necesariamente va a ser continua hasta la entrega final del proyecto.

Para finalizar esta etapa, se han generado los paquetes para su distribución tanto en Android (APK) como en Windows (UWP) para probar su funcionamiento fuera de los emuladores, exactamente en mi móvil personal y se ha entregado la aplicación básica de la Figura 3 al cliente (Entregable 2.1). Ambas aplicaciones han funcionado correctamente.

3. PLANIFICACIÓN

La tarea de planificación consiste en descomponer el trabajo a realizar en este proyecto en diferentes tareas, realizando una estimación de tiempo para cada una y siempre dependiendo de la metodología utilizada para así poder seguir una guía durante el desarrollo. Se busca principalmente cumplir los objetivos y estar preparado para cualquier imprevisto.

En principio, el desarrollo de la aplicación se va a realizar mediante 6 iteraciones las cuáles van a otorgar diferentes funcionalidades:

- Iteración 1 para la funcionalidad del acceso a la aplicación.
- Iteración 2 para la funcionalidad de la pantalla de inicio, donde se busca mostrar una agenda con los mensajes, notas y tareas.
- Iteración 3 para la funcionalidad de los pedidos.
- Iteración 4 para la de las expediciones.
- Iteración 5 para la de estadísticas, sobre todo para la gerencia de la empresa cliente.
- Iteración 6 para las funcionalidades extra de menor nivel.

1. EDT

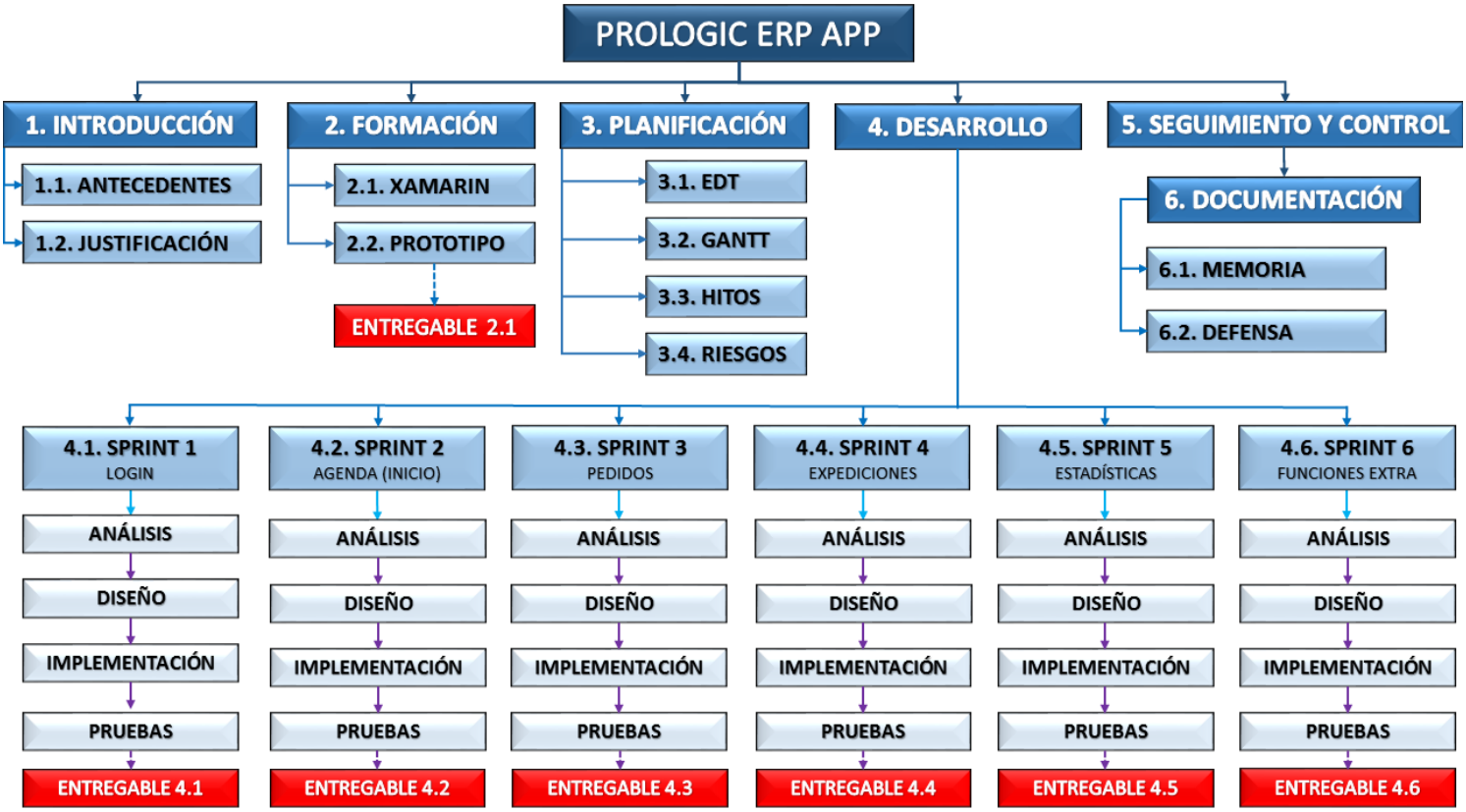

Figura 6. Estructura de descomposición de tareas del proyecto

La estructura de descomposición de tareas (Figura 6) muestra la descomposición jerárquica de las tareas que se van a llevar a cabo con sus correspondientes entregables alcanzando el objetivo final. Se utiliza principalmente para la planificación y el orden de las tareas.

1.1 Diccionario

Id	Actividad	Descripción	Fecha Inicio	Fecha Finalización
1	**Introducción**	Explicación sobre el proyecto a realizar y su justificación	**04-02-2019**	**05-02-2019**
2	**Formación**		**06-02-2019**	**20-02-2019**
2.1	**Xamarin**	Estudio de tutoriales para adquirir conocimientos sobre la herramienta.	**06-02-2019**	**11-02-2019**
2.2	**Prototipo**	Realización de varias aplicaciones básicas con los conocimientos adquiridos como prueba.	**12-02-2019**	**20-02-2019**
3	**Planificación**		**21-02-2019**	**01-03-2019**
3.1	**EDT**	Desarrollo de la estructura de descomposición de tareas.	**21-02-2019**	**22-02-2019**
3.2	**Gantt**	Desarrollo del diagrama de Gantt que muestra la planificación de las diferentes tareas con su tiempo estimado de duración.	**22-02-2019**	**26-02-2019**
3.3	**Hitos**	Desarrollo de un diagrama de hitos que muestra las fechas de los diferentes entregables del proyecto.	**26-02-2019**	**27-02-2019**
3.4	**Riesgos**	Análisis de los posibles riesgos del proyecto así como su método de prevención y minimización	**27-02-2019**	**01-03-2019**
4	**Desarrollo**	Desarrollo de la aplicación mediante iteraciones con sus correspondientes etapas y entregables.	**04-03-2019**	**24-06-2019**
5.	**Seguimiento y Control**	Etapa de control semanal de los avances del proyecto así como un seguimiento hasta el día de la entrega.	**07-02-2019**	**24-06-2019**

Tabla 1. Diccionario de la EDT del proyecto.

La estructura de descomposición de tareas va respaldada por su diccionario (Tabla 1), donde se describe detalladamente cada componente de la EDT.

En la Tabla 2 se puede ver la funcionalidad a realizar en cada iteración y el tiempo establecido para cada una, ya que no duran todas lo mismo por la importancia de la tarea a desarrollar en cada una además de que, para las primeras iteraciones, se establece un tiempo mayor del que debería por los problemas iniciales que se puedan encontrar en el proyecto. Aun así, las iteraciones 3 y 4 son las que más duran ya que es donde se espera desarrollar la funcionalidad más importante de la aplicación.

Id	Iteración	Funcionalidad	Duración
4.1	**Login**	Desarrollo de la pantalla de login y el acceso a la aplicación.	**2 semanas**
4.2	**Agenda**	Desarrollo de la pantalla inicial de la aplicación que contendrá la agenda.	**2 semanas**
4.3	**Pedidos**	Desarrollo de la funcionalidad principal que es la pantalla de pedidos.	**3 semanas**
4.4	**Expediciones**	Desarrollo de la pantalla de expediciones.	**3 semanas**
4.5	**Estadísticas**	Desarrollo de la pantalla de estadísticas.	**2 semanas**
4.6	**Funciones extra**	Desarrollo de funciones extra situadas principalmente en el menú lateral (master).	**3 semanas**

Tabla 2. Diccionario de las iteraciones del proyecto.

En cuanto a la iteración de las funciones extra, no hay una definición exacta de las funcionalidades que se van a desarrollar ya que es una iteración de prevención, es decir, una iteración que va a servir para desarrollar las funcionalidades que, tras el desarrollo del resto de las iteraciones, sean interesantes de realizar. Además, en esta iteración es donde se ha planificado desarrollar la funcionalidad que permita leer códigos QR y la compatibilidad con NFC.

Cabe destacar que la relación entre horas y semanas es 15 a 1, y la relación entre horas y días es 3 a 1, es decir, cada semana consta de 15 horas divididas en 5 días de 3 horas cada día.

2. Gantt

El diagrama de Gantt es una herramienta que ha permitido planificar el tiempo de dedicación previsto a cada tarea en la que se descompone el proyecto. Reforzando los argumentos escritos en el diccionario de la EDT para las iteraciones se puede comprobar que, incluso las que tienen establecidas el mismo tiempo de dedicación total, tienen etapas con diferente duración.

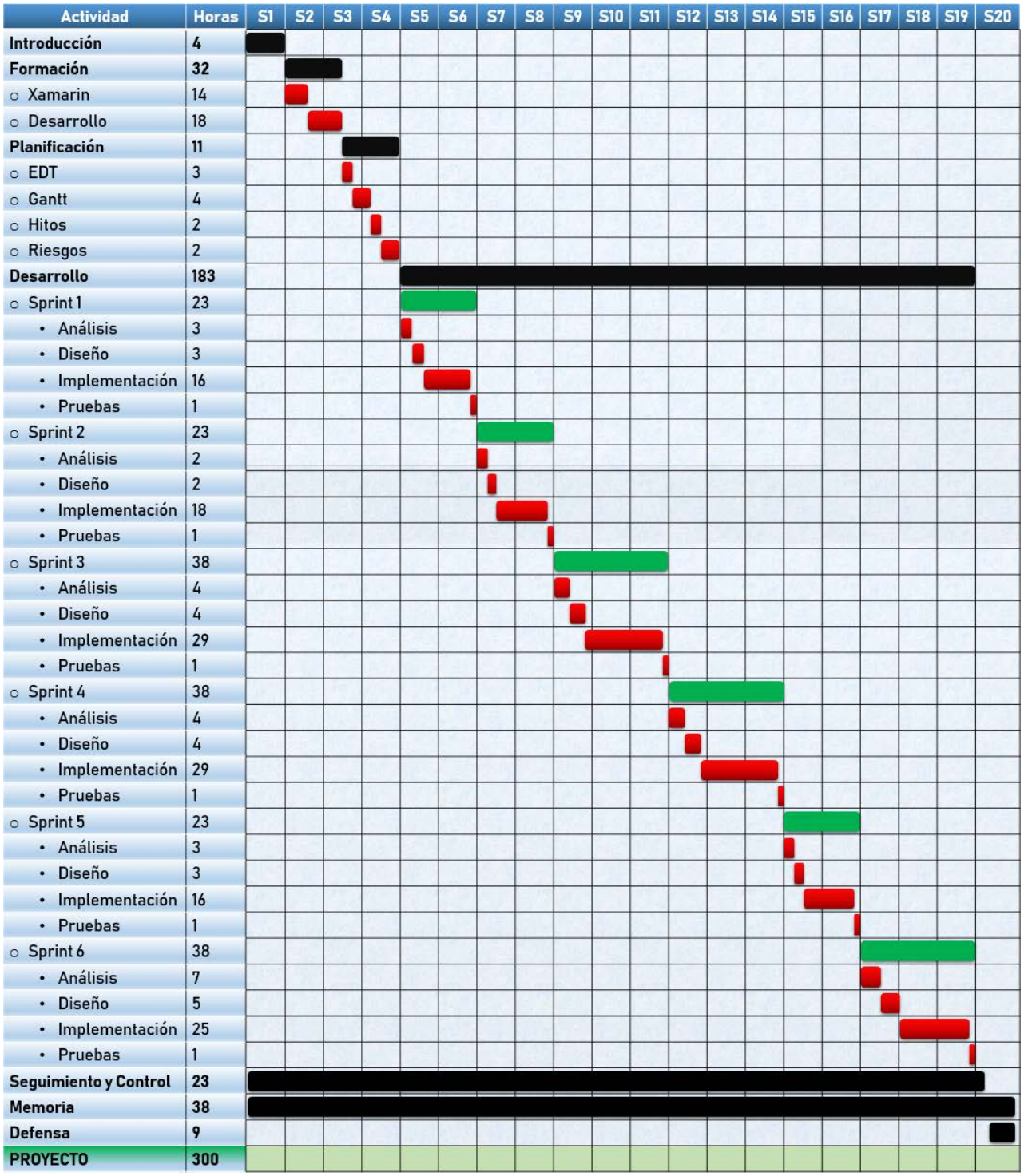

Figura 7. Diagrama de Gantt del proyecto.

Hay diferentes causas que provocan que, a pesar de que la duración es la misma, las tareas de cada iteración tengan diferente estimación (Figura 7), y son causas tanto generales para toda gestión de proyectos como particulares de este proyecto:

- Las tareas de análisis y diseño de la primera iteración suelen durar bastante más, por varias razones como los imprevistos o la ajustada planificación entre otras muchas cosas. Esta iteración es de las más simples y aun así se le ha asignado mayor duración por ser la primera.
- En la segunda iteración ya podemos normalizar la duración de cada tarea.
- La tercera y cuarta iteración tienen duraciones iguales ya que es donde se alberga toda la funcionalidad principal de la aplicación.
- La quinta iteración tiene mayor tiempo de análisis y diseño ya que necesita mayor tiempo para analizar qué datos estadísticos mostrar y de qué forma hacerlo.
- Por último, la sexta iteración, tiene una alta estimación de las tareas de análisis y diseño porque en ella se va a llevar a cabo la planificación de las funcionalidades extra, que pueden variar dependiendo del avance del proyecto y del plazo restante (por si se ha tenido que volver a planificar los plazos).

Además, hay que tener en cuenta las tareas solapadas de “Seguimiento y Control”, y “Memoria”, que aumentan el tiempo de estimación de cada tarea. Por ejemplo, la tarea de introducción consta de 4h, más el tiempo invertido en la memoria y en el seguimiento y control durante esa etapa.

3. Hitos

Dentro del proyecto se encuentran varios eventos importantes que marcan el avance del mismo. Estos eventos se corresponden con las 7 entregas parciales al cliente, 1 correspondiente con la etapa de formación y el resto con la etapa de desarrollo. Es necesario planificar una fecha de entrega para estos eventos para su correspondiente control y seguimiento de cara a cumplir los objetivos en el tiempo establecido.

Hito	Duración	Comienzo	Fin	Calendario
Entregable 2.1	7 días	14-02-2019	20-02-2019	**Febrero** S L M X J V S D 1 1 2 3 2 4 5 6 7 8 9 10 3 11 12 13 14 15 16 17 4 18 19 20 **21** 22 23 24 5 25 26 27 28
Entregable 4.1	14 días	04-03-2019	18-03-2019	**Marzo** S L M X J V S D 5 1 2 3 6 4 5 6 7 8 9 10 7 11 12 13 14 15 16 17 8 **18** 19 20 21 22 23 24 9 25 26 27 28 29 30 31
Entregable 4.2	14 días	18-03-2019	01-04-2019	**Abril** S L M X J V S D 10 **1** 2 3 4 5 6 7 11 8 9 10 11 12 13 14 12 15 16 17 18 19 20 21 13 **22** 23 24 25 26 27 28 14 29 30
Entregable 4.3	21 días	01-04-2019	22-04-2019	
Entregable 4.4	21 días	22-04-2019	13-05-2019	**Mayo** S L M X J V S D 14 1 2 3 4 5 15 6 7 8 9 10 11 12 16 **13** 14 15 16 17 18 19 17 20 21 22 23 24 25 26 18 **27** 28 29 30 31
Entregable 4.5	14 días	13-05-2019	27-05-2019	
Entregable 4.6	21 días	27-05-2019	17-06-2019	**Junio** S L M X J V S D 18 1 2 19 3 4 5 6 7 8 9 20 10 11 12 13 14 15 16 21 **17** 18 19 20 21 22 23 22 24 25 26 27 28 29 30

Tabla 3. Hitos del proyecto.

4. Plan de riesgos

El plan de riesgos es un aspecto clave a la hora de planificar este proyecto, ya que se busca su posterior comercialización entre los clientes del software Prologic ERP, por tanto, hay que tener definidos y controlados los riesgos para evitar imprevistos y fuertes impactos que impidan conseguir el objetivo en el plazo establecido.

Está orientado a los riesgos con el cliente, Ángel Tijero gerente de la empresa Logical Rioja SL, con el desarrollo y desarrollador del producto y con la tutora, Ana Romero Ibáñez profesora de la Universidad de La Rioja.

Riesgo	Actor	Prevención	Minimización	Impacto
Insatisfacción con el producto final	Cliente	Entregas parciales y comunicación directa con el cliente.	Informar al cliente de los avances y los cambios.	**MUY ALTO**
Falta de conocimientos tecnológicos	Desarrollador	Realizar una etapa previa de formación.	Ampliar plazos con el cliente.	**MUY ALTO**
Cambios en los requisitos del proyecto	Cliente	Realizar un análisis de requisitos correcto y aclarado con el cliente en cada iteración.	Mediante Scrum minimizamos este impacto.	**ALTO**
Pérdida del desarrollo	Desarrollador	Utilizar almacenamiento solapado físicamente y en la nube.	Obtener la última versión guardada	**MUY ALTO**
Plazos insuficientes	Desarrollador	Revisar la planificación con la tutora.	Modificar plazos o eliminar requisitos.	**ALTO**
Falta de comunicación con el cliente	Desarrollador/Cliente	Planificar reuniones semanales para ver los avances y cambios.	Insistir en la importancia del feedback del cliente.	**ALTO**
Falta de comunicación con la tutora	Desarrollador/Tutora	Planificar reuniones semanales para ver avances, cambios y resolver dudas.	Informar a la tutora sobre los avances del proyecto.	**ALTO**

Tabla 4. Plan de riesgos del proyecto.

4. DESARROLLO

Esta etapa recoge las diferentes iteraciones planificadas para llevar a cabo la implementación de la aplicación. Previamente se ha establecido que cada iteración constará de 4 etapas; análisis, diseño, implementación y pruebas, y posteriormente tendrá lugar la entrega del artefacto producido en cada iteración.

1. Iteración 1 - Login

La primera iteración o sprint corresponde con la implementación de la pantalla de login y el acceso a la aplicación. Para ello, se han obtenido los requisitos funcionales y no funcionales en la reunión establecida con el cliente el 28/02/2019, se han diseñado los diferentes casos de uso, se ha implementado la funcionalidad planificada, se han hecho las pruebas necesarias para examinar su funcionamiento y por último se ha entregado el artefacto resultante.

Además, en esta iteración no se ha realizado ninguna replanificación pero sí que se han reducido 3 horas del tiempo restante repartido entre las siguientes iteraciones debido a que ha ocurrido un problema con la conexión a la API y los puertos:

- La empresa sólo permite el acceso a la API si la conexión se realiza desde una red local, lo que dificulta bastante la utilización de la aplicación para los clientes. La pérdida ha sido causada porque, durante las pruebas de esta primera iteración no se conseguía realizar una conexión a la API, hasta que se llegó a la conclusión de que la causa del problema era que la conexión no estaba permitida. Comentándolo con el gerente, el encargado de la API y el administrador de redes de la empresa, se ha llegado a la conclusión de llevar a cabo el desarrollo de la aplicación (con su correspondiente depuración y prueba) desde la red local de la oficina, aplazando la apertura de los puertos o la búsqueda de una solución hasta que se finalice el proyecto, ya que el proyecto se ha desarrollado desde las propias oficinas de la empresa y no ha habido ningún problema en utilizar sólo la red local.

1.1 Análisis

El análisis de la primera iteración consiste en la recogida de los requisitos, tanto funcionales como no funcionales, del cliente en una reunión destinada para ello. Como resultado de dicha reunión, se recogieron los siguientes requisitos:

- La aplicación debe ser multiplataforma.
- La pantalla de login debe ser simple, ágil e intuitiva.
- El usuario deberá acceder con sus credenciales; correo y contraseña.
- Una vez autenticado, sólo podrá acceder a las funcionalidades a las que tenga derecho, dependiendo del nivel de permisos que tiene asignado ese usuario.

Se han elaborado los casos de uso que recogen la funcionalidad establecida para esta primera iteración.

Esta primera iteración es la más básica de todas y que tiene una implementación directa ya que se busca solamente que el usuario acceda correctamente a la aplicación.

A continuación, podemos ver el diagrama de casos de uso asociado a esta primera iteración, donde se muestra que el usuario debe iniciar sesión. Posteriormente, tendrá acceso directo a la funcionalidad que se le permita.

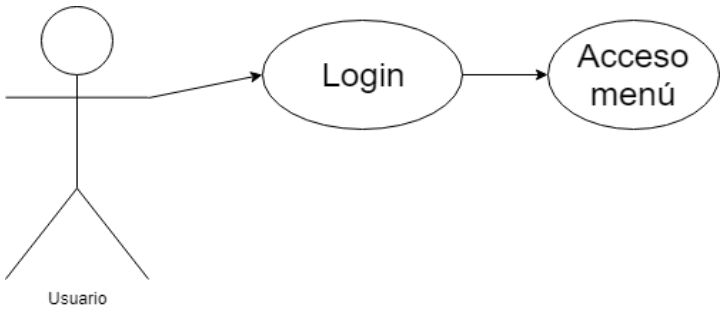

Figura 8. Diagrama de casos de uso de la Iteración 1.

1.2 Diseño

En la Figura 9 se pueden ver los prototipos del diseño de la pestaña del Login. Principalmente, al abrir la aplicación, se busca mostrar una ventana con el logo mientras se van cargando las funcionalidades de la aplicación.

Durante esta etapa de lanzamiento de la aplicación nos centramos en la carga de la funcionalidad principalmente porque está destinada a un grupo de clientes particulares que van a acceder a la aplicación para realizar ciertas tareas, es decir, van a realizar el login satisfactoriamente y por ello aprovechamos esta pequeña etapa para cargar toda la funcionalidad de la aplicación permitiendo así que, una vez iniciada la sesión, puedan realizar las tareas ágilmente.

Figura 9. Prototipo de la ventana de Login.

1.3 Implementación

Esta primera iteración la podemos dividir en dos etapas de implementación; la primera que consiste en implementar una pantalla de inicio (SplashPage) donde se carga la funcionalidad, y la segunda que consiste en implementar la pantalla de login y la carga del menú según los permisos del usuario:

- Esta aplicación tiene una funcionalidad que, sobre todo la primera vez que se inicia la aplicación, necesitará un tiempo de carga. Para cubrir este tiempo de carga, se desarrolla una pantalla de presentación denominada SplashPage en Xamarin que muestra el logo de la aplicación. Una vez cargada la funcionalidad, pasamos a la ventana del Login.
- La pantalla de Login consiste en acceso a la aplicación con un correo electrónico y contraseña asignado a cada usuario de Prologic. Para ello, la empresa Logical Rioja SL proporciona su propia API Rest [5]usada en la aplicación de pedidos web, pero que tuvo que ser modificada para adaptarse a las necesidades de la aplicación móvil. Una vez sea correcto el acceso, el usuario accederá a la pantalla principal de la aplicación.

La API Rest proporcionada por la empresa está bastante limitada a su uso en pedidos vía web, por tanto, ha sido necesario realizar varias modificaciones en ella, programada en Visual Basic:

- El acceso a la API se lleva a cabo mediante código de usuario y contraseña, mientras que la aplicación móvil busca tener un acceso mediante correo electrónico y contraseña. Para ello, se ha modificado la forma de acceder a la API.
- Según el tipo de usuario que accede a la aplicación, se debe tener acceso a ciertos apartados del menú (según los permisos), y ésta funcionalidad la API no la cubría, por tanto ha sido necesario integrarla.

Las modificaciones de la API se han realizado usando varias consultas SQL a la base de datos que almacenaba los datos de los usuarios (para hacer el login) y las ventanas a las que tenían acceso (para cargar los menús).

Cabe destacar que el funcionamiento de la API Rest está basado en el uso de tokens GUID (globally unique identifier), que funcionan como identificadores.

Existen dos tipos de tokens, el token general de acceso a la aplicación para validar los logins, y los tokens asociados a los usuarios. De este modo, para validar el acceso de un usuario se hace una petición GET pasándole en la cabecera el token general para la autenticación, y el servidor,

[5] Es una interfaz sencilla de programación de aplicaciones para obtener datos o generar operaciones que define reglas y procedimientos usada en el desarrollo de software.

en caso de que el acceso sea válido (comprobándolo en la base de datos), devuelve un objeto "Operario" (nombre utilizado en la implementación del ERP que representa un cliente, en este caso, el cliente conectado a la aplicación móvil) con su token correspondiente y otra tipo de información como, por ejemplo, su nombre completo o su correo electrónico. A partir de este token se van haciendo todas las peticiones a la API relacionadas con ese usuario (el token pasa a ser su identificación dentro de la API).

Las peticiones a la API Rest se llevan a cabo mediante JSon[6], al ser más simple, más ligero y más dinámico, sobre todo a la hora de realizar pruebas para ver si la lectura de los datos se realizaba correctamente, y para ello se ha utilizado una librería llamada Json.NET desarrollada por NewtonSoft(2019), que se ha descargado desde NuGet[7].

Para finalizar con la implementación de esta primera iteración, se ha publicado la API Rest, que es un proyecto en .NET, en un servidor web con el que se han ido haciendo las pruebas. Este servidor se ha montado en IIS Microsoft, que es un conjunto de servicios para servidores, sobre todo para servidores web, siguiendo un tutorial de Microsoft (2019).

1.4 Pruebas

Una vez finalizada la etapa de implementación, se realizan las pruebas del artefacto producido antes de ser entregado al cliente, para comprobar su correcto funcionamiento, que se adapte a las necesidades y que además no tenga ningún bug ni pueda comportarse de manera extraña, sobre todo en este tipo funcionalidades (login) que necesitan un alto nivel de seguridad.

Como se puede ver en la Figura 10, la interfaz se adapta perfectamente a lo esperado, el acceso es correcto y además se ha añadido como funcionalidad extra poder ver la contraseña antes de hacer el login, por si no se ha escrito correctamente.

Cabe destacar que la elección del tema y de los colores de la aplicación está basada en el software de escritorio y en el logo del mismo.

[6] Estándar sencillo basado en texto plano para llevar a cabo el intercambio de información.
[7] Mecanismo integrado en Visual Studio cuya misión es la creación y el intercambio de código entre desarrolladores en forma de paquetes.

Figura 10. Prueba del resultado final de la primera iteración.

2. Iteración 2 – Inicio

La segunda iteración o sprint corresponde a la implementación de la pantalla de inicio, que almacena la agenda del usuario con los mensajes, notas y tareas (a partir de ahora elementos). Para ello, se han obtenido los requisitos funcionales y no funcionales en la reunión establecida con el cliente el 19/03/2019, se han diseñado los diferentes casos de uso, se ha implementado la funcionalidad planificada, se han hecho las pruebas necesarias para examinar su funcionamiento y por último se ha entregado el artefacto resultante el día 02/04/2019, recibiendo el feedback del cliente.

Este feedback recibido ha provocado la replanificación de la iteración debido a la necesidad de incorporar nuevas funcionalidades de interés, alargando la finalización de la misma hasta el 23/04/2019, acabando con la entrega del segundo artefacto el mismo día.

2.1 Análisis

Inicialmente, para esta iteración, se planteó la idea de ser sólo de consulta, con los siguientes requisitos funcionales y no funcionales que se han recogido en la primera reunión con el cliente de esta iteración, completándose así la siguiente lista de requisitos:

- El usuario debe poder ver los mensajes que ha recibido.

- El usuario debe poder ver los mensajes que ha enviado.
- El usuario debe poder ver las notas creadas.
- El usuario debe poder ver las tareas creadas.
- Nada más acceder a la aplicación, el usuario debe poder ver todos los elementos de la agenda ordenados por tipo y hora de creación.
- Los elementos importantes o sin leer deben estar subrayados.
- Se cargarán todos los elementos de la agenda (mensajes, notas, tareas) referidos al operario activo en la pantalla inicial de la agenda.

Estos requisitos se consiguieron desarrollar para el artefacto entregado a fecha 02/04/2019, llegando a la finalización de la planificación de esta iteración pero, recibiendo feedback de parte del cliente, se llevó a cabo una replanificación de gran impacto en nuestro proyecto.

El cliente, una vez probado el entregable, se encontraba bastante contento con el trabajo realizado y pidió profundizar bastante más en la funcionalidad que se desarrolló en esta iteración de modo que quedara desarrollada al completo, y posteriormente se avanzara a la siguiente iteración.

Por tanto, se han tenido que añadir nuevos requisitos a la lista, que serían los siguientes:

- El usuario debe poder eliminar los mensajes de cualquier tipo, las notas y las tareas.
- El usuario debe poder tener un apartado de mensajes de borrador, donde almacenar los mensajes escritos pero no enviados.
- El usuario debe poder modificar los mensajes de borrador, las notas y las tareas.
- El usuario debe poder enviar los mensajes del borrador directamente.
- El usuario debe poder responder un mensaje recibido directamente.
- El usuario debe poder enviar nuevos mensajes.
- El usuario debe poder crear nuevas notas y tareas.
- El usuario debe poder iniciar y finalizar las tareas creadas.

Se puede comprobar que hay dos bloques bastante diferenciados en esta iteración; la parte anterior al feedback, donde la funcionalidad desarrollada se basaba en la consulta de la agenda, desarrollando así una interfaz intuitiva y cómoda para el usuario, y la parte posterior al feedback, donde la funcionalidad desarrollada se basaba en la modificación de la agenda, desde la creación de nuevos elementos hasta la propia eliminación de los mismos. Estos dos bloques concluyen en la funcionalidad de la agenda totalmente completada.

A continuación, finalizamos la etapa del análisis realizando los diagramas de casos de uso, separando ambos bloques debido a su desarrollo independiente a pesar de que están totalmente relacionados.

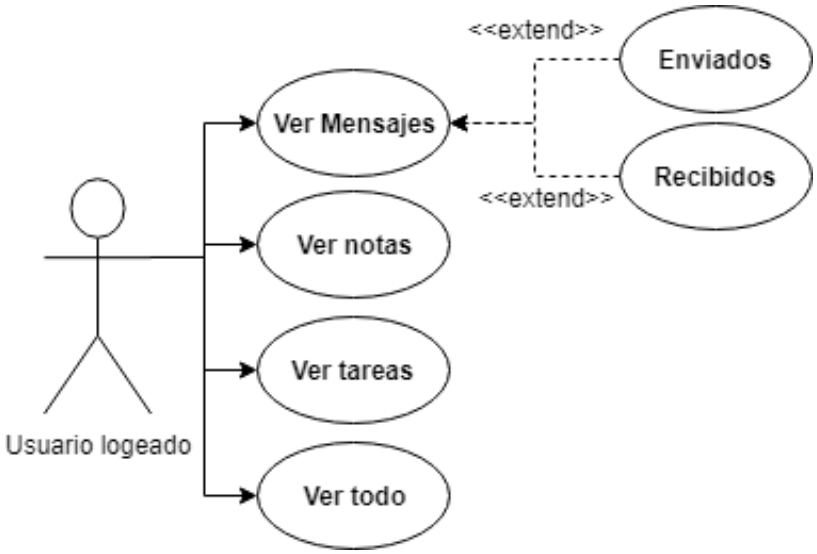

Figura 12. Diagrama de casos de uso del primer bloque de la segunda iteración (Agenda).

Respecto a la consulta de los mensajes, la idea principal que se tenía era que se pudieran ver tantos los mensajes enviados como los recibidos, y posteriormente poder filtrar y sólo ver los de un tipo u otro.

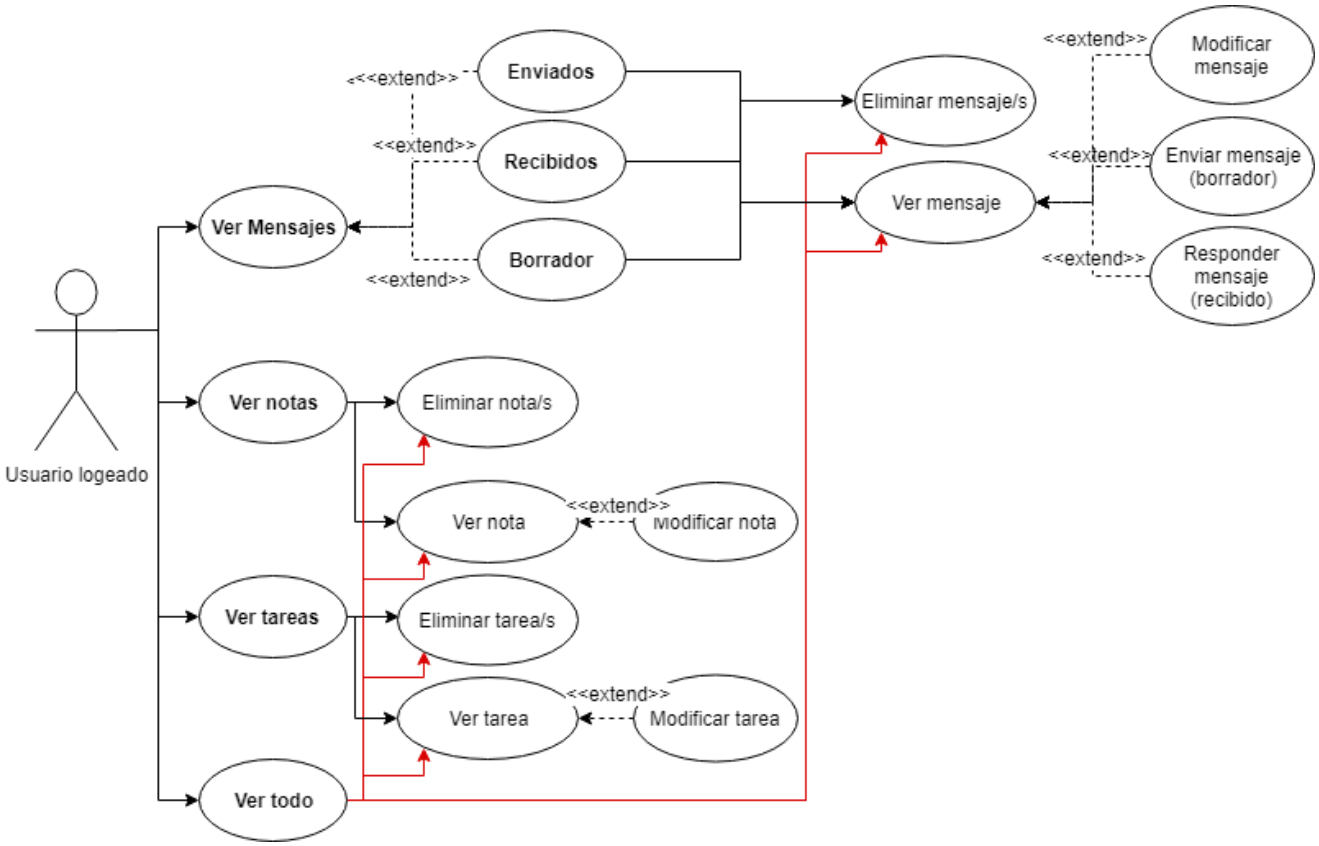

Figura 13. Diagrama de casos de uso completo de la segunda iteración (Agenda).

Como podemos comprobar, un requisito importante era que la pantalla inicial de la agenda fuera capaz de mostrar todos los elementos de la agenda ordenados por fecha y tipo, y en este caso es “Ver todo”, capaz de ver cada elemento y realizar las gestiones necesarias sobre él.

2.2 Diseño

En la Figura 14 podemos ver los prototipos de las diferentes ventanas que forman la pantalla de agenda (listado de elementos, listado de los tipos de mensajes, creación de un nuevo elemento y modificación de un elemento nuevo). Se ha considerado adecuado seguir un diseño similar basado en patrones generales de diseño de sistemas de agenda existentes bastantes potentes, como puede ser el ejemplo de Gmail y Outlook, ya que el usuario final debe tener bastantes facilidades para poder ver la información pudiendo modificarla sin tener ningún problema y, generalmente, una persona se acostumbra a usar uno de estos sistemas de agenda.

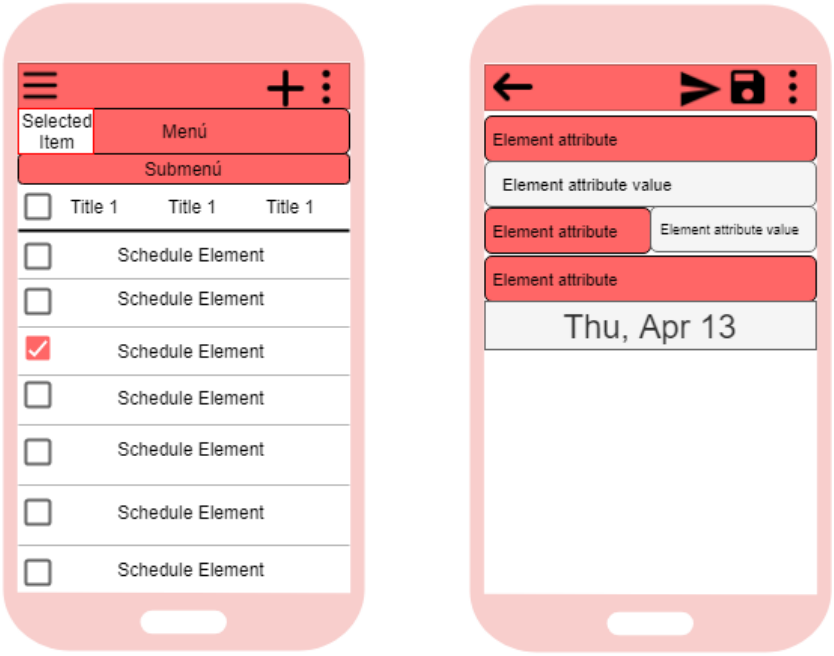

Figura 14. Prototipo de las ventanas que conforman la agenda.

El listado de elementos (prototipo de la izquierda) es igual en todos los listados, solo cambia la información a mostrar y desde el mismo listado se pueden eliminar los elementos, seleccionando todos los elementos (activando las cajas) y desplegando el menú secundario (esquina superior derecha).

Para visualizar el contenido de un elemento, modificarlo o crear uno nuevo, se accede directamente desde un listado, haciendo clic en el elemento a modificar o al icono de nuevo elemento (+). Una vez en la ventana de gestión de un elemento (prototipo de la derecha), se puede enviar el elemento (para mensajes), o guardarlo (modificaciones).

2.3 Implementación

Debido a la replanificación realizada también se va a dividir esta etapa en dos bloques bastante diferenciados como se ha establecido antes; el bloque de consulta, en el cual se ha implementado la interfaz de la agenda y la consulta a la información, y el bloque de modificación, en el cual se ha implementado toda la funcionalidad de modificación de la base de datos (creación de elementos, modificación, eliminación).

- En primer lugar, durante el desarrollo del primer bloque se ha implementado la ventana que muestra los listados de los elementos. Por un lado y siguiendo el orden, se han ido desarrollando las ventanas Todo>Mensajes>Notas>Tareas. Estas ventanas cargaban la información de la agenda que se encontraba almacenada en el objeto operario (usuario conectado) de manera que, cuando el usuario (operario activo) se conectaba a la aplicación, se validaban sus credenciales y se le asignaba la agenda (esto se ha realizado de esta manera ya que, una vez validado, se accedía directamente a la ventana de la agenda). Esta asignación la llevaba a cabo la vista del modelo de la agenda (AgendaViewModel), que se encargaba de ponerse en contacto con el modelo (su función principal ha sido la de operar con las bases de datos).
  Una vez hecha esta carga de información, se dividía el elemento mensajes en enviados y recibidos clasificándolos según los atributos “Emisor” y “Destinatario” del mensaje (en la ventana de los mensajes enviados se almacenaban los mensajes cuyo código de destinatario en la base de datos era el código del operario activo).
  En cuanto a la API, se ha desarrollado un controlador genérico denominado Agenda.ashx en el que se ha definido el método `doGet(codigoOperarioActivo)` que devolvía la información de la agenda del operario activo a partir de su código. Este detalle es muy importante para entender el funcionamiento de la aplicación: como hemos visto en la iteración anterior de acceso a la aplicación (login), si el acceso era correcto se creaba un objeto “Operario”. Este objeto corresponde con el usuario conectado, y a medida que va navegando por las diferentes ventanas, se va cargando la información necesaria asociada a ese cliente. Sobre la ventana de la agenda, cuando el usuario accedía a ella, se cargaba toda la información relacionada con la agenda en el objeto operario de manera que, mediante una visión global de la aplicación, podemos ver un objeto operario que proporciona la información a todas las vistas de la aplicación.
  Por otro lado, se ha desarrollado la ventana que mostraba la información de un elemento que recibe por parte de las ventanas desarrolladas anteriormente. Es una ventana bastante sencilla e intuitiva para que el usuario la pueda entender fácilmente (prototipo de la derecha de la Figura 14).

- En segundo lugar, durante el desarrollo del segundo bloque (bastante más extenso que el primero en cuanto a desarrollo), se han implementado todas las funcionalidades de modificación de la información de la base de datos y por ello gran parte del trabajo se ha realizado en la API. En cuanto a desarrollo de interfaz, se ha desarrollado la ventana "Nuevo" de cada elemento basada principalmente en la ventana "Ver" de cada elemento, añadiendo la funcionalidad del botón guardar para su modificación y creación (notas y tareas), del botón enviar para su creación (mensajes) y del botón eliminar para su eliminación (en las ventanas de listados). En cuanto al trabajo de la API, se han creado los controladores genéricos para la gestión de mensajes, notas y tareas. En cada uno de estos controladores se ha definido los métodos PUT, POST, Y DELETE para la correspondiente modificación, creación y modificación de elementos.

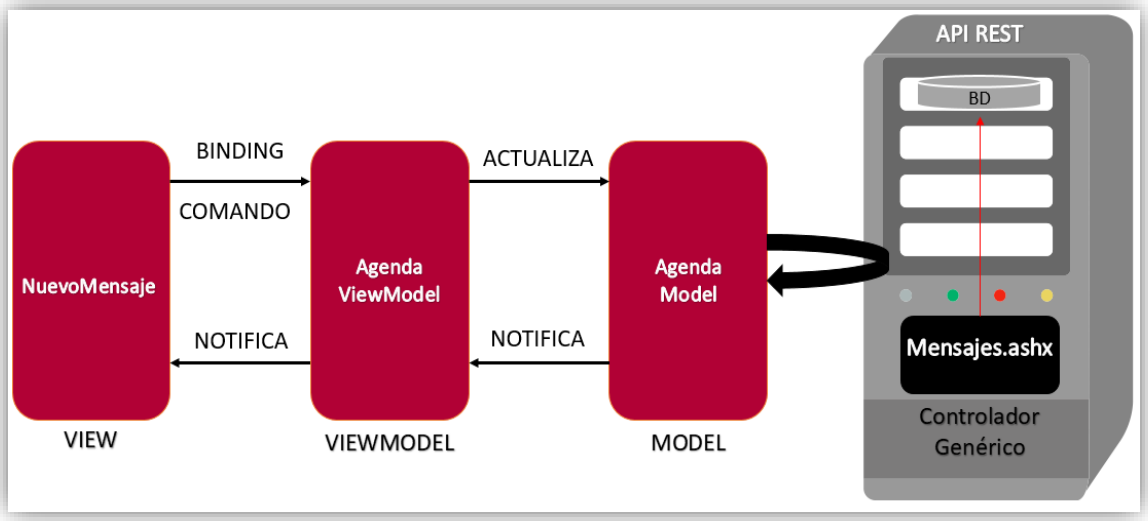

Figura 15. Patrón MVVM desarrollado para la funcionalidad de crear un mensaje nuevo.

Gracias al patrón MVVM y a los diferentes controladores genéricos se ha desarrollado una funcionalidad bastante lógica y estructurada, donde las diferentes ventanas que mostraban información accedían al View Model de la agenda, que tenía dos funciones principales; acceder al Model de la agenda que cedía la información necesaria a la base de datos, y recoger la información que éste le daba, para que las ventanas de consulta mostraran la información actualizada.

Un detalle importante para la comodidad del usuario ha sido hacer la aplicación de Windows (UWP) responsive manualmente, de manera que, cuando se minimizaba o se cambiaba el tamaño de la ventana, se cambiaba la disposición de algunos elementos (concretamente en las ventanas "Nuevo" y "Ver"), a pesar de que el cambio de tamaño lo hacía directamente Xamarin, pero no se veían bien algunos datos.

En cuanto a los iconos, se han usado iconos de licencia libre obtenidos a partir de un servicio de Google llamado Material Design (2019) el cual ofrece iconos para el desarrollo libre de aplicaciones móviles gracias a la colaboración de desarrolladores con diseñadores anónimos.

2.4 Pruebas

Una vez finalizada ya la etapa de implementación, cerramos la iteración con la etapa de pruebas, probando el correcto funcionamiento y el cumplimiento de los requisitos para posteriormente entregarle el artefacto resultante de este sprint.

Como conclusión, se puede ver que la interfaz desarrollada es bastante simple, facilitando mucho al usuario de la aplicación su uso, además de que la funcionalidad implementada carga la información en un tiempo suficiente para no ser molesto (por falta de agilidad), por tanto, se puede cerrar esta iteración y avanzar a la siguiente.

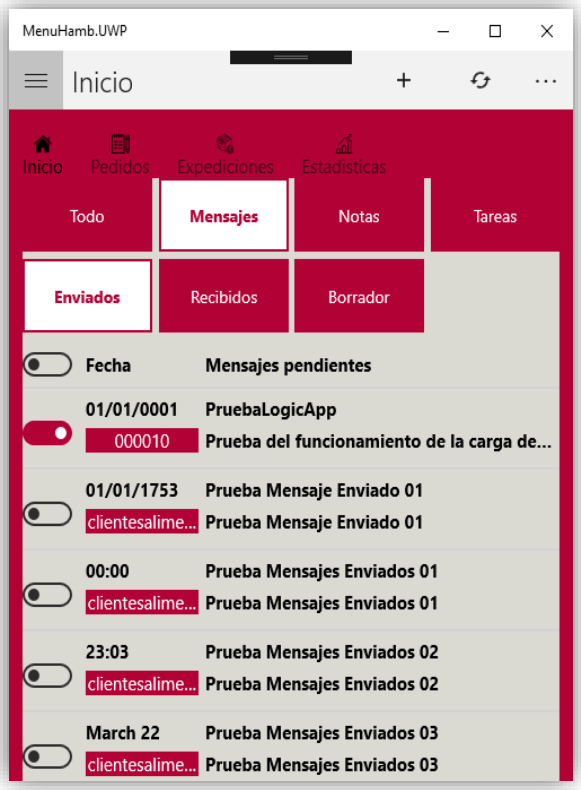

Figura 16. Ventana de mensajes enviados en UWP.

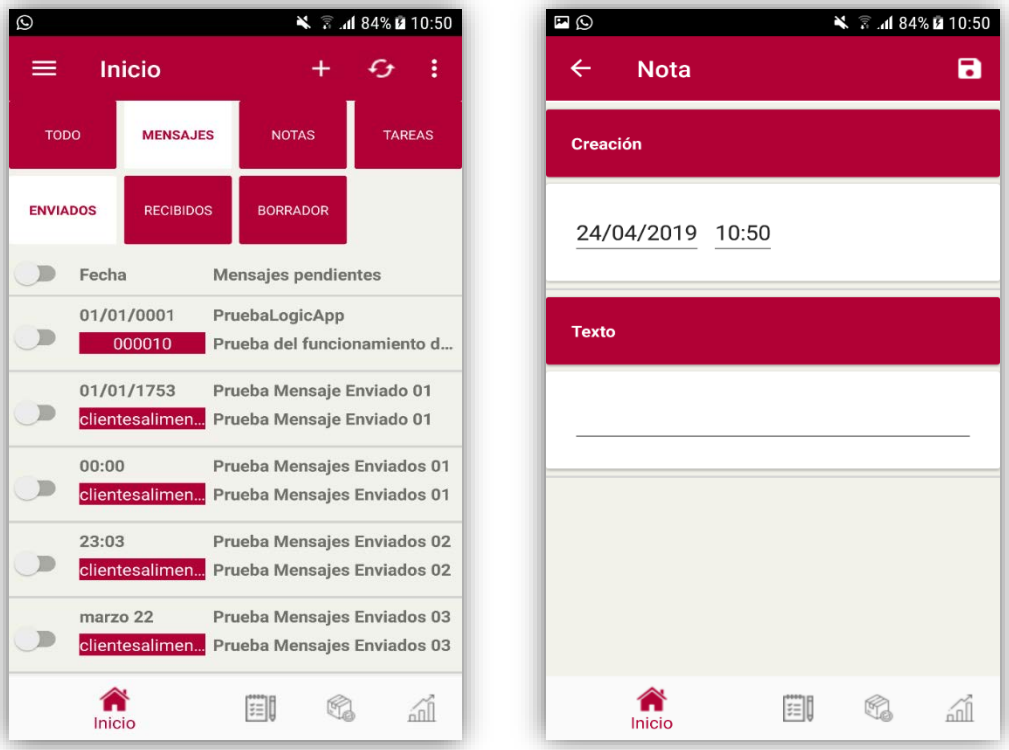

Figura 17. Prueba del funcionamiento de los mensajes enviados y su creación en Android.

3. Iteración 3 – Pedidos

La tercera iteración o sprint consiste en la implementación de la funcionalidad de la gestión de los pedidos, siendo la principal razón de la aplicación ya que, principalmente, se buscaba que los usuarios de la aplicación pudieran crear pedidos, y por ello la que más importancia requiere.

Para ello, se han obtenido los requisitos funcionales y no funcionales en la reunión establecida con el cliente el 29/04/2019, se han diseñado los diferentes casos de uso, se ha implementado la funcionalidad planificada y se han hecho las pruebas necesarias para examinar su funcionamiento.

Es importante destacar que, a parte de la replanificación del proyecto causada por la iteración anterior, también se ha realizado otra replanificación principalmente centrada en las funcionalidades a implementar en esta iteración: aunque inicialmente se había planificado una sexta iteración donde se iban a desarrollar funcionalidades extras necesarias o de apoyo para las otras funcionalidades desarrolladas en las demás iteraciones, y al hacer un análisis del tiempo restante para entregar el proyecto y las iteraciones que aún faltaban por implementar, se ha decidido implementar en esta iteración todas aquellas funcionalidades extra relacionadas con las iteraciones ya creadas, como son la posibilidad de cerrar sesión, consultar los artículos que hay (tanto activos como inactivos) y la consulta de formas de pago.

3.1 Análisis

El análisis de la tercera iteración consiste en la recogida de los requisitos del cliente, tanto funcionales como no funcionales, en una reunión destinada para ello. Como resultado de dicha reunión, se recogieron los siguientes requisitos:

- El usuario debe poder crear pedidos.
- El usuario debe poder modificar pedidos.
- El usuario debe poder eliminar pedidos.
- El usuario debe poder añadir líneas de pedido a un pedido ya existente.
- El usuario debe poder modificar líneas de pedido a un pedido ya existente.
- El usuario debe poder eliminar líneas de pedido a un pedido ya existente.

Pero, como se ha mencionado antes, se han añadido nuevos requisitos para implementar funcionalidades extra, que son los siguientes:

- El usuario debe poder cerrar sesión.
  Este requisito es necesario debido a que a los clientes les puede interesar poder cerrar sesión para acceder de nuevo con otro correo electrónico por diversas razones, sobre

todo razones relacionadas con los permisos que tiene cada usuario para ver o realizar diversas cosas o incluso para ver el correo.

- El usuario debe poder consultar todos los artículos que existen.
  Este requisito es necesario debido a que, a la hora de crear un pedido de un artículo, sólo se muestra el código y la descripción del mismo, habiendo mucha información por detrás que le puede ser de utilidad para el cliente, por tanto, se creará una ventana aparte donde poder buscar toda la información relacionada con un artículo.
- El usuario debe poder consultar todas las formas de pago que existen.
  Este requisito es necesario por el mismo argumento que el requisito anterior.

Además, en el feedback de la iteración anterior, al cliente le interesaba bastante cambiar la forma de selección de elementos eliminando así los Switchs[8] de las aplicaciones móviles (Android e iOS), elaborando así un nuevo requisito:

- En los dispositivos móviles, eliminar los Switchs para la selección de elementos, e implementar la funcionalidad que permita seleccionar elementos mediante el pulso largo sobre él.

A continuación, finalizamos la etapa de análisis realizando los diagramas de casos de uso separados según la funcionalidad a realizar, ya que aquí distinguimos perfectamente las funcionalidades de esta iteración con las funcionalidades extra.

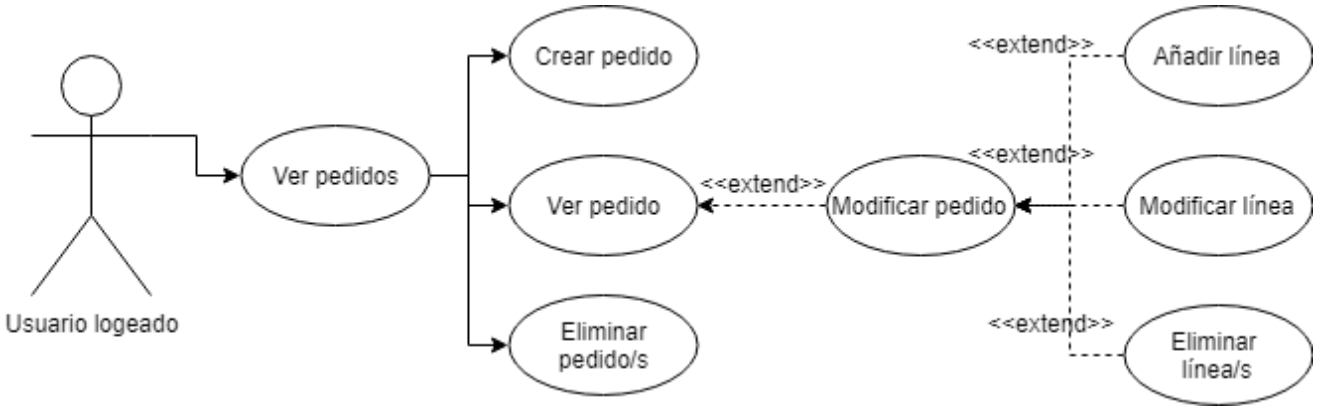

Figura 18. Diagrama de casos de uso correspondiente a la funcionalidad de los pedidos.

Para entender correctamente el diagrama de casos de uso expuesto en la Figura 18, es necesario conocer el significado subyacente de un pedido, y de su funcionamiento en el software Prologic ERP:

Un pedido está formado por un cliente, una fecha de petición, una fecha de carga, una fecha de entrega, una situación (determinada por la situación de las líneas), la forma de pago y un

[8] Permite a los usuarios alternar entre dos estados, activado y desactivado.

conjunto de líneas. Estas líneas corresponden con los artículos y están formados (en principio) por su situación, la cantidad y el coste.

Pero esta definición de líneas de pedido es bastante sencilla para lo que realmente podemos encontrar en el software ERP, ya que encontramos muchísima más información como por ejemplo el tipo de unidad (si en vez de venderse unidades sueltas, hablamos de cajas de unidades sueltas lo que hace que, entre otras cosas, el cálculo del coste de la línea sea diferente).

Por ello, la modificación de un pedido puede darse de dos formas principalmente:

- Modificación de los datos del pedido como, por ejemplo, cambio de cliente, de fechas o de forma de pago.
- Modificación de los datos de las líneas del pedido como, por ejemplo, cambio del artículo, de la cantidad o de su precio.

Es importante destacar que la funcionalidad desarrollada está relacionada únicamente con los pedidos de venta, a pesar de que el software ERP Prologic gestiona tanto los pedidos de venta como los de compra.

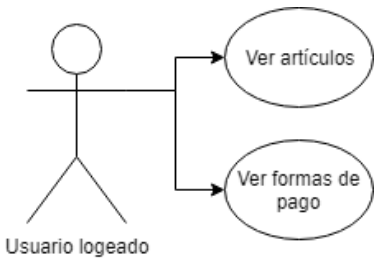

Figura 19. Diagrama de casos de uso correspondiente a la funcionalidad extra.

Finalmente, en la Figura 19 podemos ver el diagrama de casos de uso que muestra las funcionalidades extra relacionadas con los pedidos que han sido implementadas en esta iteración.

3.2 Diseño

En la Figura 20 se pueden ver los prototipos de las ventanas que conforman la pantalla de los pedidos que, de izquierda a derecha, corresponden con las ventanas de:

- Ventana de búsqueda de pedidos.
- Ventana del listado de pedidos buscado, formado por una cuadrícula.

- Ventana de ver pedido, donde se muestran sus datos y sus diferentes líneas (en una cuadrícula).

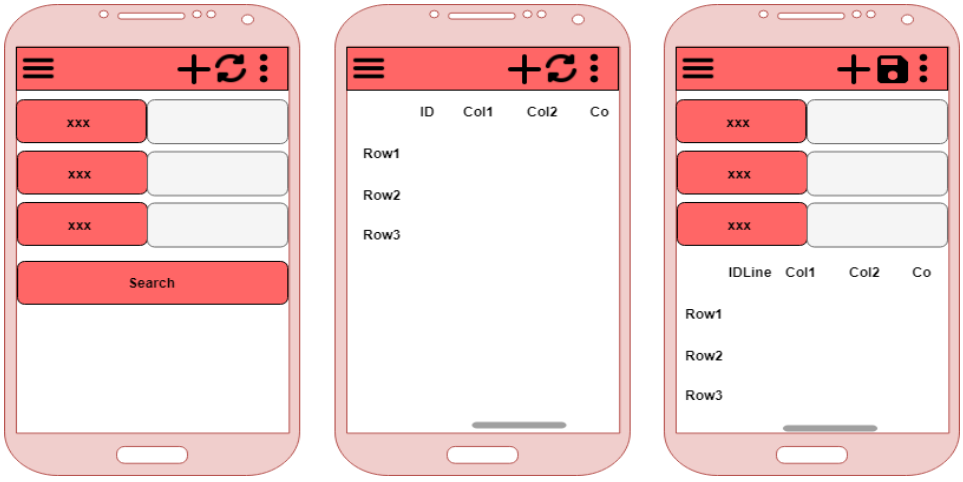

Figura 20. Prototipo de las ventanas que conforman los pedidos.

Es importante fijarse en el detalle de las barras de desplazamiento horizontal que se encuentran en las dos últimas ventanas. Estas barras nos permiten visualizar las diferentes columnas de los pedidos y las líneas de pedido en una misma ventana, pensando siempre en la comodidad y la facilidad del usuario para visualizar la información.

Figura 21. Prototipo de la funcionalidad de cerrar sesión (Clicando en el icono).

A continuación, en la Figura 21, podemos ver en la forma que tiene el cliente de poder cerrar sesión: desplegando el menú horizontal, le saldrá información sobre el usuario conectado (nombre y correo electrónico) y, clicando en el icono, podrá cerrar sesión y conectarse con otro usuario.

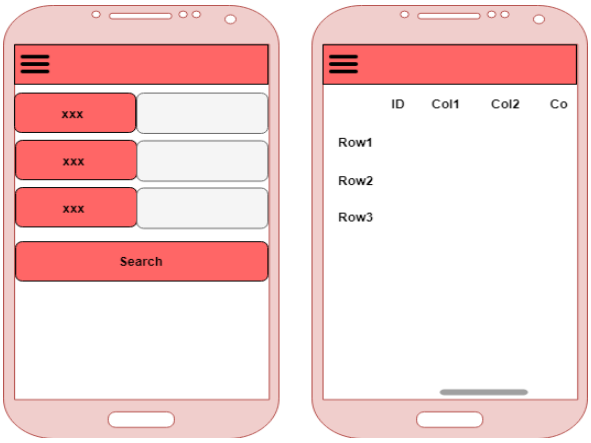

Figura 22. Prototipo de las ventanas que conforman las funcionalidades extra.

Y para terminar con la etapa de diseño de esta iteración, desarrollamos los prototipos de las ventanas de búsqueda de artículos y formas de pago (Figura 22). Estas ventanas son prácticamente iguales, solo cambian los parámetros de búsqueda, ya que son ventanas que sirven para buscar y mostrar información.

3.3 Implementación

Este apartado, como el desarrollo de la iteración anterior, también se va a dividir en dos bloques claramente diferenciados; el bloque del desarrollo de la funcionalidad de los pedidos, implementando las ventanas de búsqueda, consulta y modificación de pedidos, y el bloque del desarrollo de la funcionalidad extra, de apoyo, implementando las ventanas de búsqueda de artículos y formas de pago, y la funcionalidad de cerrar sesión.

En primer lugar, tenemos el bloque de los pedidos, conformado por las siguientes ventanas:

- Ventana de búsqueda de pedidos:

  Esta ventana está formada principalmente por los parámetros de búsqueda que son independientes de la plataforma (UWP vs Android & iOS) donde se ejecute, es decir, no se hace ninguna diferenciación entre plataformas para mostrar información diferente, solamente para la disposición de los elementos.

  Estos parámetros son los mismos que se muestran en el software Prologic ERP:

  - Año (2017, 2018,2019 o Todos*).
  - Serie (Se cargan directamente desde la API).
  - Situación (Se cargan directamente desde la API).
  - Código (Código del pedido para obtenerlo directamente).
  - Cliente (Cliente que ha realizado el pedido).
  - Artículo (filtra sólo los pedidos que contienen al menos una línea con ese artículo).
  - Fechas de pedido, carga y entrega.

  Se han definido varios parámetros que requieren información directa desde la API (Serie y Situación), y ello se consigue de la siguiente manera: cuando el cliente accede a esta ventana, se hace una petición GET a un método de la API (también desarrollado) que devuelve la información relativa a todos los pedidos, información compuesta por las Series,

las Situaciones, los Clientes activos, los Artículos activos (aunque no sean para ventas) y las formas de pago.

El software Prologic ERP tiene una herramienta llamada Infragistics que le ha permitido desarrollar una gran aplicación con prototipos muy interactivos para sus clientes. En particular, a la hora de buscar clientes, artículos o modificar el cliente de un pedido o el artículo de una línea, permite la búsqueda dinámica del elemento, de manera que, en el momento en que vas escribiendo el código del elemento (o el nombre en los clientes y la descripción en los artículos), se van mostrando los elementos que se emparejan con el texto escrito. Esta herramienta (Infragistics) no ha sido integrada en esta aplicación móvil, por tanto, la forma de asemejar este funcionamiento en la aplicación ha sido desarrollar una funcionalidad que lo permita, de la siguiente manera: mediante una barra de búsqueda (SearchBar) y un selector de elementos (Picker), se ha añadido la funcionalidad al evento de clic sobre el botón de búsqueda de la SearchBar, de manera que, cuando se reconozca un clic sobre él, se manda una petición GET a la API con el texto que ha escrito el cliente en la barra de búsqueda (diferenciando si se busca sobre un cliente o sobre un artículo), y se devuelve una lista de elementos que se emparejen con el texto escrito. Ambos métodos de la API (para clientes y para artículo), han tenido que ser desarrollados y además comprueban que se empareje con el código del cliente/artículo o con el nombre de cliente/descripción del artículo.

De esta manera, seguimos brindando al usuario con más comodidades y facilidades, siempre intentando asemejar el funcionamiento de la aplicación con el software Prologic ERP.

Se ha añadido un botón para la búsqueda (que tiene el mismo funcionamiento que el botón actualizar de la barra superior), que hace una petición GET a la API Rest con los filtros de búsqueda y que devuelve una lista con todos los objetos Pedido requeridos.

Por último, se han añadido varios botones en la barra de herramientas:

- El botón de eliminar, que hace una petición DELETE a la API con los elementos seleccionados.
- El botón actualizar, que tiene el mismo funcionamiento que el botón de búsqueda (se ha desarrollado porque en el software Prologic ERP la búsqueda se hace mediante el botón de actualizar).
- El botón de nuevo, que abre una nueva ventana de Ver Pedido pasándole como parámetro un pedido nuevo, asignándole un código “-1” hasta que éste sea guardado (más cómodo reutilizar la ventana de ver pedido, aprovechando todas las funcionalidades desarrolladas, que crear una ventana nueva de Nuevo Pedido).

En cuanto a la forma de selección de elementos para ser eliminados, se podía hacer mediante el control de Xamarin Switch pero, como se ha mencionado antes, al cliente le interesaba mucho más que la selección en las aplicaciones móviles (Android e iOS) se

hiciera con el pulso largo sobre la fila de la cuadrícula donde se encontraba el elemento. En la iteración anterior se hizo mediante Switchs ya que los cuadros de texto (Label) no tenían un evento relacionado con el pulso largo, pero en esta iteración se ha conseguido creando un Label customizado: Xamarin tiene la posibilidad de tener representadores personalizados, pudiendo así personalizar la apariencia y el comportamiento de los controles de Xamarin.Forms. De esta manera, se ha creado una nueva clase llamada CustomLabelRender que hereda de LabelRender añadiéndole un detector para el pulso largo lanzando así su evento correspondiente (también se ha realizado para el simple clic que tampoco tenía y que anteriormente se hacía añadiéndole un reconocedor de gesto de pulso al elemento). En iOS, el procedimiento es el mismo, aunque se ha tenido que hacer con los eventos propios de iOS, por tanto, ha tenido que ser escrito en un archivo aparte. Por último, se ha diseñado un modo selección, de manera que, una vez seleccionada una fila (mediante el pulso largo), se entraba en el modo selección, que consistía en poder seleccionar las demás filas mediante un simple clic pudiendo así seleccionar varios elementos de manera más ágil (ya que es algo bastante frecuente en un ERP) y poder realizar acciones sobre todas ellas (generalmente, eliminar las filas). Para salir de este modo selección y volver al normal, basta con no tener ninguna fila seleccionada.

- Ventana de ver pedido:

A esta ventana se tiene acceso haciendo clic sobre un pedido de la lista de pedidos que aparecen en la búsqueda (Ventana anterior).

Muestra la información relativa a los pedidos, y aquí sí que se hace una diferenciación de plataformas mayor que en la ventana anterior, tan grande que se ha optado por desarrollar el código de la plataforma UWP en un archivo, y el código de las plataformas móviles (Android e iOS) en otro archivo, a pesar de que gran cantidad de código era igual o muy semejante, pero por comodidad (sobre todo por la cuadrícula) no se ha escrito en el mismo archivo.

En cuanto a la exposición de la información, ha sido una tarea bastante complicada encontrar la mejor manera de hacerlo en las aplicaciones móviles, ya que en la aplicación UWP de Windows la ventana es lo suficientemente grande para poder mostrar la información sin muchos problemas. Se ha optado por realizarlo de la siguiente manera: se ha fijado la columna del identificador de la línea del pedido, y se ha incluido una barra de desplazamiento horizontal, de manera que se podía desplazar la ventana horizontalmente viendo las diferentes columnas de la cuadrícula manteniendo siempre fija la columna del identificador de la línea para no perderlo nunca de vista. Esta forma de hacerlo también ha sido incluida en la ventana de búsqueda de pedidos donde se cargaban todos los pedidos filtrados según los parámetros de búsqueda.

Por último, se han añadido varios botones en la barra de herramientas:

- El botón eliminar, que eliminaba la línea de la cuadrícula y reordenaba el identificador de las líneas. Estos cambios no se confirmaban hasta que se guardaban los cambios.
- El botón añadir, que añadía una línea al final del pedido.
- El botón guardar, que realizaba una petición PUT a la API Rest con el pedido que estaba activo, donde se aprovechaba para añadir o eliminar las filas. Para ello, se ha desarrollado en la API un método PUT que funcionaba de la siguiente manera: este método recibía un objeto pedido a actualizar. Si había cambios en la cabecera (información del pedido), se realizaban los cambios. A parte, se comprobaban las líneas que conformaban este pedido de manera que, si existía una línea que no estaba en el pedido original, se añadía porque se entendida que era una línea nueva, y si existía una línea que en el pedido original que ya no estaba en el pedido modificado, se eliminaba porque se entendía que esta había sido eliminada.
  De esta manera, no hacía falta tener que desarrollar un controlador genérico para los objetos LineaPedido.

Es bastante importante realizar varios controles antes de poder guardar la modificación de un pedido en la aplicación:

- La cantidad no podía ser 0, sino saltaba una excepción en la API, así que mejor evitarlo desde la aplicación.
- No podía crearse una línea sin artículo, por tanto, antes de guardar, se comprobaba que todas las líneas tenían artículo, y en caso de existir alguna que no tuviera, se paraba la finalización y se mostraba un mensaje para que el usuario modificase de nuevo los datos erróneos.
- Todos los artículos que se mostraban en el Picker de selección de artículos (como se ha explicado antes para asemejarse al funcionamiento de Infragistics en el software Prologic ERP) tenían que ser artículos que puedan ser vendidos. Para ello, se eliminaban de los artículos disponibles aquellos que no se pudiesen vender.

En cuanto a la implementación de la funcionalidad extra, la dividimos en tres partes:

- Cerrar sesión: la funcionalidad de cerrar sesión se ha hecho de una manera bastante sencilla (gracias a la forma en la que se ha ido desarrollando la aplicación), añadiendo un icono, el nombre del operario activo y su correo electrónico (de acceso) en el menú lateral desplegable. El icono tiene un aspa roja, que cuando recibe un clic se ejecuta la funcionalidad de cierre de sesión. Esta funcionalidad simplemente borra el parámetro de aplicación “Operario” que haya activo y redirecciona al cliente a la pantalla de inicio de sesión. Una vez que el usuario inicie de nuevo sesión, se pisará cualquier información antes cargada.

- Mostrar artículo y forma de pago: Ambas funcionalidades también han sido bastante sencillas después de haber desarrollado varias ventanas de consulta. Cada una de ellas muestra un formulario de búsqueda con los parámetros necesarios para cada tipo, y clicando sobre los elementos que aparecen en el listado de búsqueda se redirige al cliente a las ventanas Ver Artículo y Ver Forma de Pago, mostrando la información necesaria en cada uno.

3.4 Pruebas

Una vez finalizada ya la etapa de implementación, cerramos la iteración con la etapa de pruebas, probando el correcto funcionamiento y el cumplimiento de los requisitos para posteriormente entregarle el artefacto resultante de este sprint. Debido a las diversas funcionalidades aquí implementadas, se han realizado varias pruebas profundas sobre todo por la delicadeza de la funcionalidad de la gestión de los pedidos.

En primer lugar, en la Figura 23, nos encontramos con la ventana de pedidos, donde se cargan todos los pedidos asociados al operario que se encuentra activo (usuario conectado). Como se puede ver, en la parte posterior se encuentra el formulario de búsqueda de pedidos, donde podemos filtrar por año, serie, situación, código, cliente asociado, artículo (sería necesaria con que el pedido tuviese una línea con ese artículo) y las diferentes fechas. Desde esta misma ventana, se pueden crear nuevos pedidos (+), actualizar los datos por si hubiera habido algún cambio mientras se estaba consultado la información y eliminar pedidos.

Una vez se haya seleccionado un pedido para su visualización (mediante un clic sobre la fila del pedido), se accede a la ventana mostrada en la Figura 24, donde se puede ver la información relativa al pedido (llamada cabecera) y el conjunto de líneas que lo componen. De esta manera, definimos un pedido como un conjunto de cabecera + líneas. En cuanto a la cabecera, se puede modificar el cliente del pedido, la forma de pago y las diferentes fechas y, para las líneas, se puede modificar el artículo que la forma, su descripción, la cantidad, y el precio del artículo. Cabe destacar que la modificación de los datos relacionados con el artículo solo afectaría al mismo dentro de esta línea, y que el artículo en sí no va a ser modificado.

En cuanto a la creación de nuevos elementos, al hacer clic en el botón nuevo (+) de la Figura 23 se muestra un nuevo pedido vacío cuya única información disponible es el código, -1 (indicando que es un código nuevo y que no está guardado) que corresponde a la ventana que se aprecia en la Figura 24. Y, al hacer clic en el botón nuevo (+) de la Figura 24, se añade una nueva línea debajo de la última, asignándole una nueva situación por defecto y un código de línea. A partir de ahí, el usuario puede modificar los datos, rellenar los campos vacíos a su gusto y guardar los cambios (siempre y cuando cumplan con las condiciones).

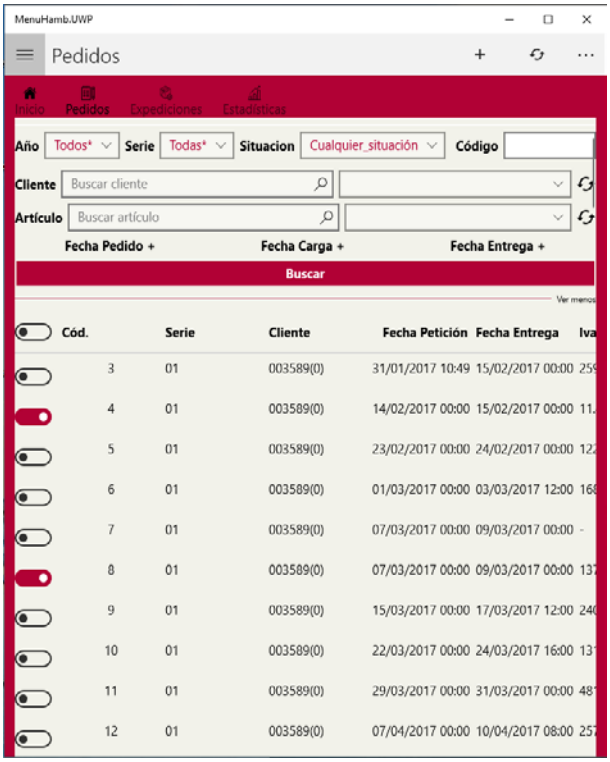

Figura 23. Ventana de búsqueda y listado de pedidos en la aplicación UWP.

En la Figura 24, se pueden ver dos detalles muy importantes (para las aplicaciones móviles): en primer lugar, separando la pantalla entre la cabecera y las líneas encontramos un Label cuyo texto es "Ver menos". Su funcionalidad es simple pero bastante interesante, ya que se encarga de esconder la cabecera si el usuario lo ve necesario, sobre todo cuando hay demasiadas líneas, para así poder ver únicamente las líneas en toda la ventana. Una vez escondida la cabecera, el texto del Label cambiará a "Ver más" y, al hacer clic en él, se volverá al estado inicial. Por otro lado tenemos una barra de desplazamiento lateral que, manteniendo la columna del código de línea fijo, permite al usuario desplazarse horizontalmente para ver toda la información relativa a la línea del pedido. De esta manera, se ha conseguido mostrar la gran información que tiene un pedido y una línea de pedido de forma cómoda para el usuario.

Figura 24. Ventana de ver pedido en la aplicación para Android.

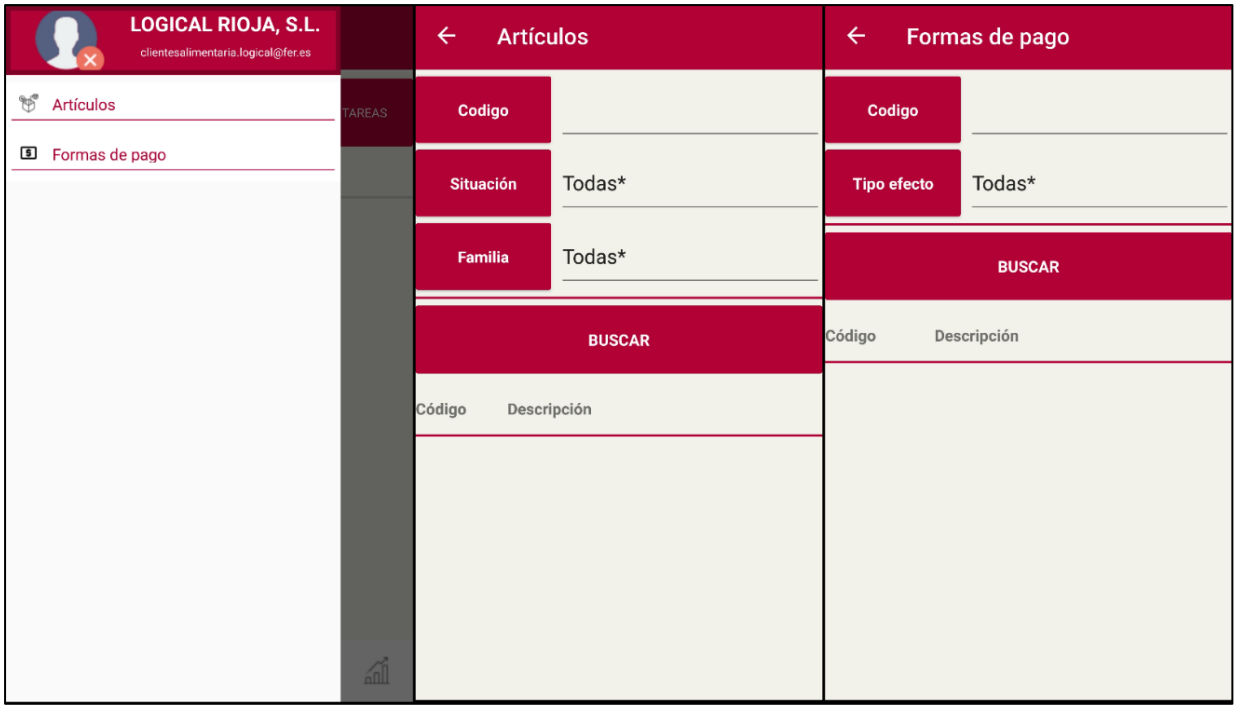

Figura 25. Interfaces para la consulta de artículos y formas de pago.

También se han llevado a cabo las pruebas relacionadas con la funcionalidad extra desarrollada, y en la Figura 25 se puede ver la interfaz para la consulta de artículos y formas de pago, que funcionan correctamente. La funcionalidad restante de cerrar sesión también ha sido probada y no ha dado ningún problema.

Tras recibir el feedback del cliente, el cual sólo ha pedido realizar un cambio (que todos los precios sean alineados a la derecha), se ha podido cerrar correctamente esta iteración, ya que se ha hecho lo esperado y la interfaz ha sido valorada satisfactoriamente por el cliente.

Una vez finalizada esta iteración, y teniendo en cuenta el retraso sufrido respecto a la planificación inicial al por la ampliación de los requisitos en las iteraciones 2 y 3, se ha decidido no realizar las iteraciones 4 y 5 correspondientes al desarrollo de la funcionalidad de las expediciones y las estadísticas.

Por otro lado, era necesario llevar a cabo un profundo control de la funcionalidad desarrollada hasta ahora, comprobando que, tras las pruebas realizadas en esta iteración, no saltase ninguna excepción en la aplicación causada por el intercambio de información con la API. En una reunión establecida con la tutora, se ha tomado la decisión de incluir una nueva iteración no planificada en el proyecto en la cual se llevara a cabo el seguimiento y control de la aplicación desarrollada hasta ese momento, principalmente para el control de las excepciones. De esta manera surge una nueva iteración extra, la 4.

4. Iteración 4 – Control de excepciones

La cuarta iteración o sprint, llamada también iteración extra, consiste en la implementación de la funcionalidad que controle y gestione las excepciones que puedan surgir tras el trabajo realizado en las iteraciones anteriores, provocadas principalmente por el intercambio de información con la API.

4.1 Análisis

Tras las pruebas realizadas en la iteración anterior, se ha comprobado que en la API podían surgir determinados problemas afectando directamente a la aplicación, provocando fallos en ella o un descontrol en su comportamiento. De ahí nace la necesidad de controlar las excepciones para que, en determinados momentos que saltan excepciones, evitar que la aplicación entre en modo interrupción (no responde) o responda con un mensaje de error grave y que el usuario no pueda continuar con la ejecución. Estas excepciones pueden ser verdaderos problemas que necesitan la supervisión de un programador o simplemente pequeños errores no esperados que no tenían que haber parado la ejecución de la aplicación, sino que con haber controlado la excepción hubiera sido suficiente.

4.2 Diseño

Durante el desarrollo de este proyecto, han ido saliendo muchas “zonas críticas”, es decir, parte del código susceptible de lanzar una excepción, sobre todo tras la tercera iteración. En unos casos, la excepción podía ser lanzada por culpa de la aplicación, pero la mayoría de veces estaba causada por la API, por tanto, era necesario llevar un control de excepciones, tanto en la aplicación como en la API.

La idea principal es tener varias capas de control de excepciones, estructuradas de la siguiente manera:

- En las zonas críticas, rodear la zona mediante bloques try-catch, capturando la excepción esperada y realizando las respectivas gestiones.
- En las funcionalidades que haya varias zonas críticas o zonas propensas a lanzar excepciones, rodear la zona mediante un bloque try-catch, capturando las excepciones inesperadas y analizando si, con la información recibida, se puede continuar con la ejecución, y en el caso de que esto no fuera así, mostrar un mensaje de error y redirigir al usuario a otra ventana o impedirle realizar la acción, pero en cualquier caso evitar que se cierre la aplicación. Sobre todo, se usa esto para los objetos que almacenan información que proviene directamente de la API (por ejemplo, los pedidos

relacionados con el operario, puede ser nulo si ha habido algún problema en la API), por tanto, en algunos momentos también se comprueba que este no sea nulo.

- En cada plataforma (Android, iOS y UWP), debe haber una capa superior por donde van a pasar todas las excepciones lanzadas que no han sido controladas de manera que, cualquier excepción que salte y no haya sido gestionada por ningún bloque try-catch pero sí lanzada, que sea capturada en esta capa y que evite el cierre sorpresivo de la aplicación. De esta manera, evitaríamos que la aplicación actuase de un modo inesperado, e informaríamos al usuario en todo momento de si ha habido algún problema, muy importante en este ámbito que se gestiona información que afecta directamente a los clientes.

4.3 Implementación

La implementación de esta iteración se ha basado principalmente en ver las zonas críticas de las iteraciones anteriores y buscarles una solución. El resultado ha sido el siguiente:

- Una zona crítica bastante importante era la carga de los elementos de la agenda, es decir, de los tipos de mensajes, las notas y las tareas. Esta carga provenía directamente de la API envuelta en un objeto operario y, debido a muchas causas diferentes, esta información podía ser nula. Por tanto, la solución más eficaz era comprobar que esta no era nula, y en caso de lo que fuera, cargar una agenda que estuviera vacía, mostrando el mensaje correspondiente (“Agenda vacía”).
- Para el envío de mensajes, se cargaba la lista de contactos asociados al operario activo y también era información que venía directamente de la API por tanto en alguna situación podría ser nula. La solución a esto ha sido evitar que se enviara cualquier mensaje si no había ningún contacto cargado, ya que no existiría ningún destinatario.
- En cuanto a los pedidos, que es la funcionalidad que más control ha necesitado, se ha llevado a cabo un intenso trabajo en la API, eliminando cualquier suspicacia de devolver un nulo, aunque sea, devolviendo elementos vacíos. Esto se debe por varias razones: la funcionalidad de los pedidos conlleva la carga de información relativa a los pedidos (no la carga de pedidos, sino información sobre pedidos, como por ejemplo, las series que hay o las diferentes situaciones de un pedido, para poder filtrar la búsqueda) y esta podría ser nula en algún momento (por culpa de la API), por tanto, para evitar que sea nula, devolver una lista vacía (de las situaciones de un pedido por ejemplo) y en la aplicación encargarse de que no se cargara ningún pedido si la información de los pedidos no era la correcta. Por tanto, aquí nos encontramos con un ejemplo de control en el que, si la información no es la correcta, lo mejor es comunicárselo al cliente e impedir que siga ejecutando la acción requerida. El mismo ejemplo nos sirve para la carga de artículos.

Por otro lado, teníamos la modificación de pedidos, tanto de las cabeceras como de las líneas. El usuario tenía el permiso de introducir la información que él quisiese, por ejemplo, introducir "quince" en cantidad excepto de 15, por tanto, ha sido algo que se ha tenido que controlar, tanto para la cantidad como para cualquier control de tipo Editor (llamado así al control de Xamarin que permitía introducir información) y, además, se les ha asignado como zonas críticas porque en algún momento podrían lanzar excepciones no esperadas.

Por otro lado, para controlar las excepciones de la API, se ha creado una clase llamada "Error", compuesta por un código, un enunciado y una descripción. Se encargaba de recoger las respuestas de la API y, en caso de que el código del objeto error correspondiese con uno de los códigos asociados a las excepciones (como se pueden ver en la tabla 5), se mostraba un mensaje de error y se impedía continuar con la ejecución de la operación solicitada. La mayoría de veces este objeto error es capaz de proporcionar la suficiente información al usuario para que pudiese arreglar el error por sí mismo (por ejemplo, formato de datos introducidos incorrecto) y el resto de veces proporcionaba la información necesaria al programador para conocer el error y poder encontrar una solución.

Código	Descripción	Información adicional
0	Ejecución correcta.	
1	No autorizado.	
2	Configuración errónea del usuario autorizado.	
100	Verbo HTTP no admitido.	
101	Acción solicitada no válida.	
102	Parámetros URL no válidos para la acción solicitada.	
103	Cuerpo de la petición no válido para la acción solicitada.	
200	Datos no encontrados.	
201	Datos ya existentes.	
202	Error de validación de datos.	El campo data del error contendrá las causas del error.
203	Datos no guardados.	
404	Conexión a internet no disponible.	
9998	Excepción no gestionada.	
9999	Error inesperado.	

Tabla 5. Códigos de error de la aplicación.

4.4 Pruebas

Una vez finalizada la etapa de implementación, se realizan las pruebas de cada capa de gestión de excepciones, intentando recrear todos los escenarios posibles que puedan provocar la aparición de una excepción

En primer lugar, se han hecho pruebas en el envío de información entre la aplicación y la API, sobre todo en el envío por parte de la API. Para realizar estas pruebas, se han creado escenarios en el que la API devolviese información nula, comprobando que se mostraba un mensaje de error en la aplicación y que se impedían realizar ciertas funciones, pero evitando en todo momento un cierre sorpresivo. Un ejemplo de ello sería el siguiente:

- La API devuelve una agenda nula por diversas razones, por ejemplo, porque no se han encontrado tareas relacionadas con el operario activo. En ese caso, cuando el usuario accede a ver los mensajes, se encuentra con el mensaje de “Agenda vacía”.

En segundo lugar, se han hecho pruebas en la introducción de datos por parte del cliente, sobre todo en los pedidos, ya que en la agenda se podía introducir cualquier tipo de carácter alfanumérico y símbolos. Un ejemplo de ello sería el siguiente:

- Si el usuario introduce letras en vez de números en la cantidad de artículos en una línea de pedido, esta era leída y mostrada como un “0”, y además se controlaba la excepción que pudiera surgir al intentar convertir una letra en un número. Al intentar actualizar cualquier línea con información errónea, los mecanismos de control de la API lanzarían una excepción que, en la aplicación móvil se asociaría con un objeto Error, y se mostraría la información correspondiente a este error, dando pistas al usuario de cuál ha sido el error (Error 202 de la Tabla 5).

Por último, se han hecho pruebas del funcionamiento de la capa superior, creando escenarios ficticios en el que se lanzara una excepción (en principio esperada, pero lanzada como si fuera una excepción inesperada únicamente para comprobar su funcionamiento), y se han controlado correctamente. Es importante destacar que en esa última capa solo se intenta captar las excepciones que no han sido tratadas en ningún momento, sino que han sido lanzadas por bloques de control.

5. SEGUIMIENTO & CONTROL

En esta fase se ha realizado un seguimiento de todas las tareas establecidas en la planificación para su control en torno al cumplimiento de los objetivos y los plazos. Con ello, se busca ver el progreso del proyecto para tomar cualquier decisión correctiva necesaria.

1. Duración real de las tareas

Id	Actividad	Fecha Inicio	Fecha Finalización	Desvío
1	**Introducción**	**04-02-2019**	**05-02-2019**	Ninguno
2	**Formación**	**06-02-2019**	**20-02-2019**	Ninguno
2.1	**Xamarin**	**06-02-2019**	**11-02-2019**	Ninguno
2.2	**Desarrollo**	**12-02-2019**	**20-02-2019**	Ninguno
3	**Planificación**	**21-02-2019**	**28-02-2019**	**-3h**
3.1	**EDT**	**21-02-2019**	**22-02-2019**	Ninguno
3.2	**Gantt**	**25-02-2019**	**26-02-2019**	Ninguno
3.3	**Hitos**	**27-02-2019**	**27-02-2019**	**-3h**
3.4	**Riesgos**	**28-02-2019**	**28-02-2019**	
4	**Desarrollo**	**04-03-2019**	**20-06-2019**	**Ninguno**
4.1	**Sprint 1**	**04-03-2019**	**19-03-2019**	**+3h**
4.2	**Sprint 2**	**19-03-2019**	**26-04-2019**	**+75h**
4.3	**Sprint 3**	**29-04-2019**	**31-05-2019**	**+36h**
4.4	**Sprint Extra**	**03-06-2019**	**20-06-2019**	**+54h**

4.5	**Sprint 4**	**-**	**-**	**-63h**
4.6	**Sprint 5**	**-**	**-**	**-42h**
4.7	**Sprint 6**	**-**	**-**	**-63h**
5	**Seguimiento y control**	**04-02-2019**	**20-06-2019**	Ninguno
6	**Documentación**	**04-02-2019**	**20-06-2019**	Ninguno

Tabla 6. Duración real de las tareas del proyecto.

Las tareas planificadas pueden tener desvíos, clasificados en negativos, cuyo desvío provoca una reducción en los plazos o un aumento de tiempo para otras tareas, ya que la tarea ha durado menos de lo esperado, y en positivos, cuyo desvío provoca un aumento de plazos o una reducción de tiempo para otras tareas, ya que la tarea ha durado más de lo establecido, llegando a impulsar una replanificación.

Para realizar un seguimiento y control más eficaz, se ha decidido realizar puntos de control antes de cerrar las diferentes tareas, donde se realizaban comparativas del trabajo esperado con el trabajo realizado, y el tiempo estimado con la duración real de la tarea, además de realizar un seguimiento semanal de la situación del proyecto. En la tabla anterior podemos ver la duración real de las tareas de cada etapa.

1.1 Introducción

Esta etapa consistía en la redacción en la memoria de una introducción al proyecto a realizar. Es una etapa de poco trabajo cuya planificación ha sido bastante sencilla y exacta, y no ha habido ningún cambio que afecte a la planificación del proyecto.

1.2 Formación

Esta etapa consistía en una etapa de formación previa a la implementación de la aplicación, sobre todo para poder entrar en contacto con las tecnologías a utilizar y poder realizar una planificación más exacta del tiempo que podría costar llevar a cabo las diferentes iteraciones. Se han generado varios productos, sobre todo para visualizar el funcionamiento final de una aplicación desarrollada en Xamarin y además se ha entregado un artefacto (prototipo de la aplicación), cuyo feedback no ha provocado ningún cambio ya que el cliente estaba bastante satisfecho con dicho prototipo.

1.3 Planificación

En esta tarea se han realizado varios análisis en cuanto al trabajo a realizar y el tiempo restante para finalizar el proyecto, estableciendo los plazos y se han desarrollado varios diagramas. En cuanto a la duración planificada para esta etapa, se ha conseguido terminar un día antes de lo esperado gracias a sus últimas dos etapas (hitos y riesgos), que no han necesitado todo el tiempo esperado. De esta manera, se ha comenzado con la fase de desarrollo un día antes de lo previsto.

1.4 Desarrollo

En la etapa de desarrollo se han llevado a cabo puntos de control antes de cerrar cada iteración desarrollada, por tanto, se van a tratar independientemente:

- Iteración 1: Esta primera iteración, de entre todas la que se han cerrado, ha sido la que más se ha adaptado a la planificación. Sólo se ha perdido un día de lo que realmente se había planificado (sin contar el día extra que se ha ganado en la tarea de planificación), es decir, 3 horas de trabajo que en un proyecto de 300 horas no provoca ningún impacto de gran calibre y no ha sido necesaria hacer ninguna replanificación. Como se ha comentado anteriormente, la pérdida ha sido principalmente causada por la conexión a la API y los puertos de acceso.
- Iteración 2: La segunda iteración ha sido sin duda la que más ha afectado al proyecto, ya que ha provocado una replanificación muy significativa, pero, aparte de las causas que se explican a continuación, es algo normal porque las primeras iteraciones son las que más cuestan siempre y en esta segunda iteración es donde se ha entrado de lleno en el diseño de la aplicación y el desarrollo de la API. En esta iteración se han perdido 25 días en cuanto a la duración estimada de la tarea que, hablando en horas, son 75 horas y en un proyecto de 300 horas corresponde al 25% de la duración, por tanto, es un porcentaje bastante alto que se ha notado en el cierre del proyecto. Las causas de esta replanificación han sido varias, pero todas por el feedback del cliente: en primera instancia, el cliente quería una pantalla de agenda que sirviese de consulta, pero, después de enseñarle el artefacto resultante, quería ir un paso más allá y que, la funcionalidad de consulta, pasara a ser una funcionalidad de gestión, pudiendo además crear, modificar y eliminar elementos. A pesar de haber tenido un gran impacto, ha permitido adelantar el trabajo de otras iteraciones ya que en las siguientes iteraciones sí que se esperaba que la funcionalidad fuese más de gestión (no solo de consulta de información), por tanto, para las siguientes iteraciones ya se tenía una base para desarrollar la funcionalidad de gestión correspondiente.
- Iteración 3: La tercera iteración también ha tenido una replanificación que ha provocado un impacto en el proyecto, pero no por haber implementado cosas no

esperadas para el proyecto, ni por la materialización de un riesgo, sino porque, además de lo planificado para esta iteración, se han desarrollado funcionalidades que estaban planificadas en otras iteraciones, por tanto, la replanificación ha afectado a la duración de esta tarea pero no a la duración del proyecto, ya que la duración de las otras iteraciones donde se encontraban las funcionalidades aquí desarrolladas se ha disminuido. Entrando más en detalle, la replanificación ha sido causada por lo cerca que estaba la fecha de cierre del proyecto y, para dejar bien cerrada esta iteración que consistía en el desarrollo de la funcionalidad que controlaba la gestión de los pedidos, había funcionalidades extra planificadas para la última iteración que eran bastante interesantes de incorporar en esta iteración (exactamente, la consulta de los artículos y las formas de pago) mientras que había otras que eran necesarias incorporar antes de cerrar el proyecto (poder cerrar sesión), y por ello se ha aprovechado para desarrollarlas. Además, la última iteración también estaba planificada para las distintas funcionalidades necesarias a desarrollar que podían surgir a lo largo del desarrollo pero, como no ha dado tiempo a llevarla a cabo, se ha movido las funcionalidades necesarias que iban surgiendo a esta iteración.

- Iteración 4 (Extra): En cuanto a esta última iteración, se ha incluido en el proyecto como una iteración de control de excepciones, que ha permitido realizar un control de todas las funcionalidades desarrolladas y, por tanto, ha sido una tarea completamente de seguimiento y control del proyecto que se ha llevado a cabo hasta el último día del proyecto.

A pesar de estas replanificaciones y cambios de requisitos que han afectado bastante al proyecto, sí se ha conseguido desarrollar la parte más importante de la aplicación para el cliente, la gestión de los pedidos.

Finalmente, los problemas que se han ido teniendo durante el desarrollo de este proyecto, se han solucionado cambiando la forma de hacerlo, buscando en el foro de Xamarin(2019) o en StackOverFlow(2019), y gracias al equipo del departamento de programación de la empresa.

6. CONCLUSIONES

En cuanto a las conclusiones que podemos sacar de este proyecto, nos encontramos con la duda de si realmente se han cumplido los objetivos, si el trabajo realizado ha sido lo suficiente bueno para obtener la satisfacción por parte del cliente y las cosas que se han podido aprender, sobre todo al realizarlo en un ámbito de trabajo real, donde existía un cliente que esperaba la realización de este proyecto.

6.1 Objetivos alcanzados

Al principio de este proyecto se establecieron varios objetivos, pero había uno que estaba por encima de los demás, ya que era el que daba mayor sentido y justificaba la petición de este proyecto. Ese objetivo correspondía con la necesidad de desarrollar una aplicación móvil que pudiese permitir la gestión de los pedidos de manera más ágil y rápida, sin necesidad de tener un ordenador o un dispositivo similar a mano, sino que simplemente se pudiera llevar a cabo desde un móvil, una Tablet o incluso un PDA. Como resultado final de este proyecto, se puede concluir que ese objetivo se ha cumplido a la perfección, además de cumplir otros objetivos de bastante interés (agenda) desarrollando funcionalidades más allá de las que se habían planificado, destacando que toda la funcionalidad desarrollada ha sido desarrollada al completo para así, a la hora de continuar con el proyecto, el desarrollo se centre en nuevas funcionalidades.

6.2 Continuación del proyecto

Como ya se ha comentado varias veces, el proyecto es real y se busca su posterior comercialización entre los propios clientes del ERP Prologic, por tanto, es un proyecto que no acaba aquí. Ciertamente es que el objetivo principal se ha cumplido, pero hay varios objetivos necesarios (planteados inicialmente y no desarrollados) que se espera cerrar en un futuro próximo y así poder acabar con este proyecto, produciendo una aplicación móvil que permita también gestionar las expediciones y mostrar datos estadísticos a los gerentes de las empresas.

Además, dentro de la funcionalidad que ya se ha realizado, se puede incluir muchas mejoras más como, por ejemplo, incluir más información relativa a los pedidos y que esta pueda ser modificada. Para añadir información, únicamente habría que añadir nuevas columnas y obtener esta información a partir de la API, de la misma manera que se ha conseguido obtener la información que ya se muestra, pero, para modificarla, habría que añadir nuevos mecanismos de control para esta información. Por tanto, hay bastantes mejoras que se pueden llevar a cabo y que aumentaría la capacidad de la aplicación de asemejarse al ERP Prologic.

6.3 Lecciones aprendidas

Este proyecto se comenzó con apenas conocimientos sobre desarrollo de aplicaciones móviles, y los únicos conocimientos que había estaban relacionados con las aplicaciones de escritorio, por tanto, eran conocimiento que eran prácticamente innecesarios para este proyecto, aunque se conocía bastante bien el funcionamiento de Visual Studio, lo que ha facilitado bastantes cosas.

Dejando a un lado la tecnología, este proyecto ha permitido trabajar en un proyecto real, con su correspondiente etapa de planificación, seguimiento y control, permitiendo desarrollar un gran conocimiento sobre estas tareas, sobre todo mostrando que muchas veces se ha de realizar una replanificación, y esta ha de hacerse correctamente.

En conclusión, nos encontramos con la siguiente lista de lecciones aprendidas:

- Lo más importante, Xamarin, aprendiendo a desarrollar aplicaciones multiplataforma mediante la reutilización de código y usando el patrón MVVM.
- Inglés, ya que bastante información relativa a los problemas y soluciones que han ido apareciendo durante el desarrollo estaban en inglés.
- Capacidad de actuar en momentos críticos, cuando algo no sale de una forma, buscar una solución aparte y no estancarse.
- Realizar las replanificaciones correctamente, ya que, en caso contrario, no se hubiera cumplido el objetivo principal del proyecto.
- Tener reuniones constantes con el cliente de la aplicación, para no realizar trabajo innecesario que luego haya que cambiar o incluso eliminar.
- Tener reuniones constantes con la tutora, sobre todo para el correcto control de la planificación.

7. BIBLIOGRAFÍA

[1] Microsoft. (2019). Introducción a Xamarin.Forms. Consultado el 11 de febrero de 2019, en https://docs.microsoft.com/es-es/xamarin/get-started/index

[2] Newtonsoft. (2019). Librería JSon.NET Consultado el 07 de marzo de 2019, en https://docs.microsoft.com/es-es/xamarin/get-started/index

[3] Microsoft. (2019). Introducción a Xamarin.Forms. Consultado el 06 de marzo de 2019, en https://support.microsoft.com/es-es/help/323972/how-to-set-up-your-first-iis-web-site

[4] Xamarin Forms. (2019). Foro de Xamarin.Forms. Consultado el 12 de febrero de 2019, en https://forums.xamarin.com/categories/xamarin-forms

[5] StackOverFlow. (2019). Foro de StackOverFlow. Consultado el 12 de febrero de 2019, en https://es.stackoverflow.com/

[6] Material Icons. (2019). Iconos de código abierto. Consultado el 15 de febrero de 2019, en https://material.io/