UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

APLICACIÓN MULTIPLATAFORMA PARA LA TOMA DE COMANDAS AUTOMATIZADA EN RESTAURANTES Y CAFETERÍAS

Rosa Folgado Alufre

Tutor: Francisco José Martínez Zaldívar

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2018-19

Valencia, 1 de abril de 2019

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

Aplicación multiplataforma para la toma de comandas automatizada en restaurantes y cafeterías, de manera que se podrán realizar comandas desde el dispositivo móvil del camarero, con el objetivo de optimizar y agilizar las operaciones de atención al cliente.

Resum

Aplicació multi plataforma per agafar comandes automatitzada en restaurants i cafeteries, de manera que es podran realitzar comandes des de el dispositiu mòbil del cambrer, amb l'objectiu d'optimitzar i agilitzar les operacions d'atenció al client.

Abstract

Multiplatform application which would allow restaurants and cafeterias taking orders directly from the server's mobile device. This system allows a faster and more efficient costumer care.

Palabras clave

Aplicación multiplataforma, Restaurante, Cafetería, Xamarin, Azure, C#

Key words

Multiplatform application, Restaurant, Cafeteria, Xamarin, Azure, C#

Índice

Capítulo 1. Introducción y objetivos

Los *smartphones* llegaron para quedarse, y esta revolución digital también está ganando territorio en el mundo de la hostelería. La automatización de comandas a través del móvil permite obtener mayor rentabilidad y facilitar el trabajo a los camareros, lo que posibilita un servicio más rápido y eficaz.

España es el país europeo con más densidad mundial de lugares de hostelería (más de 270.000 restaurantes, bares, cafetería y otros establecimientos que integran el sector de la restauración), concretamente 277.539 establecimientos según los datos del Directorio Central de Empresas (DIRCE) publicados por el Instituto Nacional de Estadística (INE) en el año 2017 [1]. En la hostelería española, la búsqueda e implementación de procesos automáticos sigue siendo un campo en desarrollo y expansión [2]. La apuesta por esta digitalización se justifica por sus notables beneficios: permite ahorrar tiempo y dinero al contribuir a una gestión más eficiente para el hostelero y, a la par, favorece al cliente proporcionándole un servicio más rápido.

El proceso habitual en la mayoría de los locales de restauración sigue los siguientes pasos: se toman las comandas con boli y papel, los camareros cada vez que toman nota a una mesa, vuelven dentro a la barra o cocina para poner en marcha el pedido, pasarlo a la TPV y volver a las salas o terraza a seguir tomando nota. En cambio, el proceso digitalizado se beneficia que hoy en día cualquiera tiene un dispositivo móvil. Tan solo instalándose una aplicación ya podría disponer de un comandero digital, transformando así todo el proceso: la aplicación tendría comunicación directa con cocina, por lo que el camarero no tiene que volver a barra o cocina, simplemente llevar y recoger bandejas. Este ahorro de tiempo al marchar platos favorecería un servicio más rápido y, por tanto, mayor rotación de mesas. A su vez, facilitaría la actualización de los productos disponibles (automáticamente el camarero podría saber si un producto se ha agotado, o si puede ofrecerse un nuevo producto) y el acceso inmediato a información relevante a la hora de personalizar los pedidos (alérgenos, modificaciones del plato, el registro del punto de la carne…). Además de todo esto, las rectificaciones o modificaciones pueden aplicarse rápidamente sobre el mismo pedido, evitando repeticiones u omisiones y sus consecuentes pérdidas económicas.

Considero que la falta de digitalización de los procesos en hostelería supone un innecesario *hándicap*, que retrasa y dificulta la labor de tan extendido gremio en nuestro país. La era tecnológica en la que nos encontramos posibilita que dispositivos de tan fácil acceso como un *smartphone* se conviertan en la herramienta de trabajo que simplifique y rentabilice las funciones de este sector.

Por este motivo, el objetivo principal de este proyecto será diseñar y desarrollar una aplicación multiplataforma para facilitar y agilizar la toma de comandas en el sector de la restauración.

Los objetivos específicos se relacionan con las funciones de la aplicación y se concretan en el desarrollo de una aplicación móvil que permita, de manera dinámica:

- Tener un control de las mesas, así como del estado e importe sus comandas.

- Registrar las comandas diarias, los traspasos entre mesas y el tipo de pago efectuado. Esto podría permitir en un futuro, extraer estadísticas (productos más demandados, mesas más concurridas…) permitiendo mejorar las ventas del establecimiento.

Capítulo 2. Motivación

La programación y conocer cómo se construiría una aplicación completa, ha sido siempre un ámbito que ha captado mi atención. Tras cursar Aplicaciones Telemáticas, asignatura en la que desarrollé una sencilla aplicación Android en Java para la toma de comandas, decidí que me resultaría muy interesante ampliar el proyecto e intentar sacarle el máximo potencial.

Mi prioridad fue llevar a cabo una aplicación de toma de comandas de manera automatizada, que fuera rápida y segura, pero sobre todo, que el desarrollo fuera un reto para mí y me permitiera conocer nuevos lenguajes de programación y herramientas con las que no había trabajado.

Examinando las diferentes opciones para hacer una aplicación multiplataforma, encontré Xamarin, que ofrece un desarrollo integrado en la herramienta de desarrollo de Microsoft, Visual Studio, y que se desarrolla en lenguaje C# para .NET Framework, generando código nativo para dispositivos móviles Android, iOS y Windows Apps [3]. Ésta me pareció una herramienta de desarrollo cuyas posibilidades se ajustaban bastante a lo que necesitaba. Al mismo tiempo, se encuentra integrada dentro del programa Visual Studio, un conjunto completo de herramientas de desarrollo en el mismo entorno de desarrollo integrado (IDE), que habilita el uso compartido de herramientas y facilita la creación de soluciones en varios lenguajes. Con la versión 2017 que integra la suite de herramientas de Azure y Xamarin, es más fácil y ágil para los desarrolladores la compilación, conexión y ajuste a aplicaciones móviles [4].

Igualmente, también suscitó mucho interés en mí el lenguaje de programación, C#, un lenguaje a la alza y por el que están apostando los programadores desde hace algunos años. Está incluido en .NET Framework, que es un entorno de ejecución administrado para Windows que proporciona diversos servicios a las aplicaciones en ejecución. La biblioteca de clases de .NET Framework está orientada a objetos, proporciona una biblioteca de código probado y reutilizable al que pueden llamar los desarrolladores desde sus propias aplicaciones. Hay varios lenguajes disponibles, como Visual Basic, C#, F# y C++/CLI, de Microsoft [5].

Capítulo 3. Gestión del proyecto

Respecto a la metodología para la ejecución del proyecto, encontramos dos opciones: por un lado, las metodologías ágiles como Skrum o Programación Extrema (XP); y por otro lado las tradicionales. Al ser un proyecto individual y no ser una solución de excesiva complejidad de análisis, se ha optado por un desarrollo evolutivo o en cascada, siguiendo las metodologías tradicionales. En éstas, cada etapa del desarrollo representa una unidad y testeo, es decir, cada etapa se inicia tan pronto como ha finalizado la anterior.

Para el desarrollo se trabajará con la herramienta "Git" [6], integrada en el programa Visual Studio. Se trata de un repositorio remoto donde se subirán los cambios para tener una copia del desarrollo de la aplicación, y no solo la que se desarrolla en local.

Dividiremos la rama principal *master*, en subramas según las tareas, y cuando todo funcione y se haya testeado, se integrará con la rama principal. De esta manera, tendremos acceso al historial de cambios del código, consiguiendo una copia de *backup* cada vez que subamos nuevos cambios. También nos permitirá hacer un "*Pull Request*" de manera que, si queremos unir una de las ramas a la principal, se podría requerir la revisión del código por una segunda persona para asegurarse de la calidad del mismo o que no haya incongruencia en la estructura.

3.1 Distribución en tareas

En primer lugar, hay una etapa de análisis y toma de requisitos en la que, teniendo en cuenta el objetivo a alcanzar, se pensarán las posibles soluciones e imprescindibles que tiene que tener nuestra aplicación.

Una vez tenemos claro el qué, dónde y cuándo, pasaremos a pensar el cómo. Por lo que, empezamos la fase de diseño de la arquitectura. Elegiremos de todos los posibles productos software, el conveniente para nuestra solución y tendremos en cuenta las piezas que van a formar el proyecto.

Seguidamente, una vez tenemos claras las herramientas y programas a utilizar, y sabiendo el flujo que va a tener nuestra aplicación, llega el momento de empezar a desarrollarla.

Capítulo 4. Especificación del sistema

Este proyecto se basa en el desarrollo de una aplicación que permitirá tomar comandas de forma automatizada, de manera que el camarero con su dispositivo móvil pueda elegir la mesa y seleccionar la cantidad de cada producto.

4.1 Descripción de la solución

La aplicación estará formada por tres proyectos: por un lado, una Web Api que opera contra una base de datos SQL Server y una Web de administración, recursos desplegados en Azure; por otro lado, la solución de la aplicación multiplataforma en Xamarin.

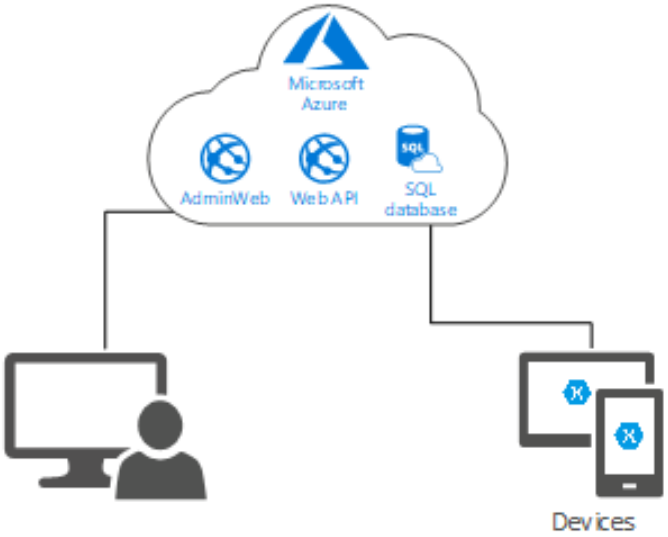

Figura 1. **Esquema solución.**

4.2 Herramientas y software utilizado

A continuación, se comentará las herramientas software empleadas durante las distintas fases de la implementación de la aplicación, ya sean "Frameworks", librerías o entornos de desarrollo.

- Visual Studio Professional
- NuGet Microsoft.Net.Http
- NuGet Newtonsoft.Json
- Framework Net Standard 4.5
- Framework Net Core 2.1
- Xamarin.Forms v.3.0.0.296286-pre2
- ASP.NET Core Web Application
- SQL Server Management Studio 17.8.1
- Entity Framework
- Microsoft Azure Platform
- Application Insights.ApsNetCore2.1

En las próximas líneas se describe la arquitectura utilizada para implantar la solución del sistema de toma de comandas automatizadas en restaurantes y cafeterías.

En primer lugar, se decide construir una arquitectura DDD (Domain Driven Design), por lo que cada proyecto se divide en capas (infraestructura, interfaz de usuario, aplicación y dominio) [7]. De este modo se facilita evitar las dependencias y desacoplar lo máximo posible el software. A continuación se describen en más detalle las capas a desarrollar.

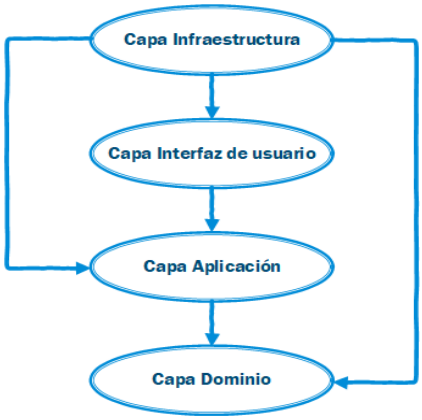

Figura 4. Esquema arquitectura DDD.

La primera capa de presentación o "Frontend", ha sido desarrollada en Xamarin Forms como un proyecto de tipo "SharedProject" [8], y la web de administración en ASP.NET, que ambas siguen una estructura MVC (Modelo Vista Controlador).

En cuanto a la web de administración, localizada en esta misma capa de presentación, se ha tratado de seguir la misma línea de desarrollo y usar Visual Studio. Al igual que el resto de proyectos de la solución, se ha utilizado ASP.NET, un entorno para aplicaciones web desarrollado por Microsoft, empleado por programadores y diseñadores para construir webs dinámicas [9]. Como se ha mencionado, se ha seguido el patrón de arquitectura MVC (Modelo Vista Controlador) puesto que es más fácil gestionar la web si la tenemos dividida en estos tres pilares [10]. También supone una ventaja a la hora de escalar porque se pueden reutilizar componentes, es decir, si fuera necesario, podríamos generar distintas vistas para un mismo modelo, gracias a este desacoplamiento del modelo respecto de la vista. Además, permite tener control sobre las URL, pudiendo formatearlas a URL semánticas.

La segunda, la capa de negocio o "Backend", incorpora toda la lógica de negocio y ha sido desarrollada como una Web API altamente modularizada en distintos servicios, según el modelo del sistema. De esta forma, el fallo de una de las operaciones de un servicio, no conlleva un fallo en el resto de los servicios (ni siquiera en el resto de las operaciones de este). Estos servicios atacan la base de datos por medio de un ORM (Object-Relational Mapping), Entity Framework [11]. El uso de un ORM es para añadir una capa extra de abstracción, de forma que, en el desarrollo, ya no se realizan consultas a tablas, sino que se trabajan directamente con Objetos equivalentes a las entidades de la base de datos. Las ventajas de este sistema, es la simplificación del desarrollo una vez es superada la curva de aprendizaje, y sobre todo, un aumento en la mantenibilidad y claridad del código, además de protección directa contra ataques de tipo "SQL Injection".

En este sentido, al ser una herramienta de Microsoft es más sencillo integrar una SQL Server a nuestra Web API que será la que maneje el BackEnd. Respecto a esto, una de las ventajas de tener nuestra base de datos SQL en la nube, es que el servicio es totalmente administrado y eso ofrece mayor compatibilidad con el motor de SQL Server. También proporciona mayor rendimiento y protección de los datos [12], ya que las conexiones siempre van encriptadas, y se supervisa la actividad para detectar amenazas potencialmente peligrosas, pudiendo avisar e intervenir lo antes posible.

En caso que la aplicación llegara a millones de dispositivos, esta base de datos se podría autoescalar (de manera vertical u horizontal, según las necesidades), mejorando el rendimiento y la latencia en grandes cargas de trabajo. Igualmente nos proporcionaría alta disponibilidad integrada, la posibilidad de hacer copias de seguridad automatizadas o la replicación geográfica para impedir que cualquier error u operación de mantenimiento pare las operaciones de negocio.

En segundo lugar, para que la aplicación sea multiplataforma, se utilizará Xamarin, una herramienta donde se puede escribir en lenguaje C# y el mismo código será traducido para su ejecución en iOS, Android y Windows Phone.

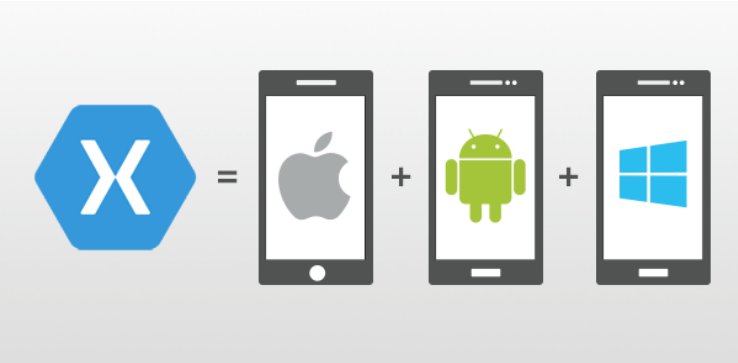

Figura 2. **Xamarin para programar en las tres plataformas.**

La ventaja de esta herramienta es que posibilita, desde un lenguaje unificado, hacer aplicaciones tanto para iOS, que crea sus apps en Objective-C; como para Android, que se programan en Java. Aunque se deben tener conocimientos básicos de cada lenguaje nativo para adaptar funciones más específicas, en general facilita mucho los procesos elementales. Esto se debe a que no es necesario escribir la misma funcionalidad en cada uno de los lenguajes de programación.

En definitiva, la optimización del código con esta herramienta, pudiendo compilar como código nativo para cada plataforma el código C# compartido, posiblemente representa el futuro del desarrollo móvil con esta nueva tecnología.

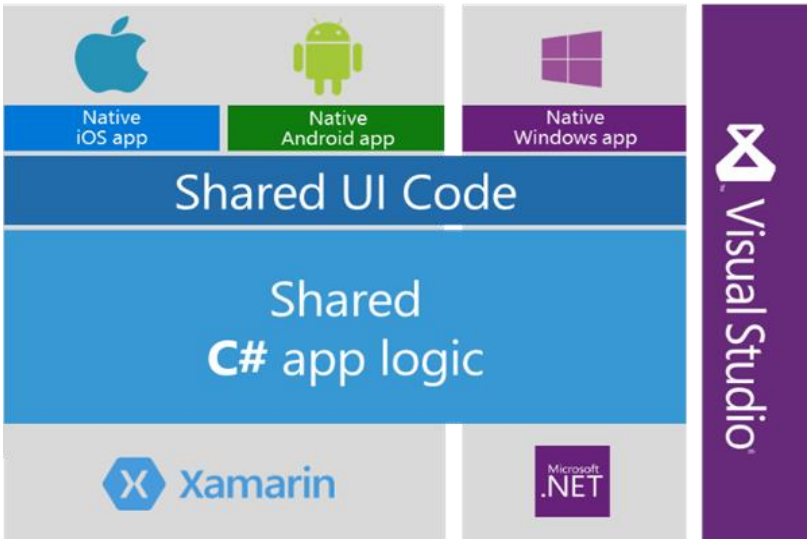

Figura 3. **Código compartido en las aplicaciones multiplataforma de Xamarin.**

Capítulo 5. Análisis y diseño

5.1 Análisis

La fase de análisis del proyecto se dedicará a especificar los requisitos tanto funcionales, como no funcionales que tiene que satisfacer el sistema y a identificar los distintos tipos de actores que existen, así como el modo en que se relaciona cada uno de estos con los requisitos o cómo interactúa con el sistema definido.

5.1.1 Requisitos

Los requisitos funcionales que tendrá que satisfacer el sistema serán:

(1) Los usuarios podrán autenticarse y crear nuevas comandas en relación con el número de mesa.

(2) La experiencia de usuario debe ser intuitiva y fácil de utilizar, teniendo los menos clicks posibles y evitando la sobrecarga de pantallas.

(3) Permitirá el acceso al resumen de la comanda para repasarla.

(4) Permitirá la administración de usuarios, productos y mesas, pudiéndose modificar, añadir o eliminar.

5.1.2 Actores

Considerando el comportamiento de la aplicación se induce un único tipo de actor: el actor general. Éste será un usuario que normalmente utilice la aplicación. Se autentica con sus credenciales (previamente dadas de alta) y podrá generar nuevas comandas, modificarlas o cerrarlas.

Sin embargo, en el caso de la web de administración, habrá tres niveles de acceso: (1) los usuarios que no tienen acceso a la web, (2) los usuarios con permisos para entrar en la web y permiso de lectura de usuarios y lectura/escritura para productos y mesas y (3) el usuario propietario, que además de tener acceso a la web será el que tenga permiso de escritura (crear, modificar o eliminar) de usuarios.

5.1.3 Casos de uso

En los distintos casos de uso se definirán las interacciones del actor con la aplicación.

<table>
<tr><th colspan="2">Inicio de sesión</th></tr>
<tr><td colspan="2">Actores:
Usuario (iniciador)</td></tr>
<tr><td colspan="2">Descripción:
El caso de uso permite al usuario autenticarse en el sistema, con el propósito de tener acceso a las funciones de acuerdo a su perfil.</td></tr>
<tr><td colspan="2">Precondiciones:
-</td></tr>
<tr><th colspan="2">Secuencia de eventos típica</th></tr>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td>1. Usuario comienza la aplicación

3. Usuario introduce sus credenciales</td><td>2. Sistema permite introducir credenciales (DNI y contraseña).

4. Sistema comprueba si las credenciales del Usuario son correctas. Si las credenciales son correctas, Usuario queda autenticado.

5. Sistema muestra nueva pantalla con cuatro botones: Tomar comanda, Traspasar mesa, Unir mesa y Pagar.</td></tr>
<tr><th colspan="2">Secuencia alternativa</th></tr>
<tr><td>4. Si las credenciales no son correctas, le aparece un mensaje de error.</td><td></td></tr>
<tr><td colspan="2">Postcondiciones:
El usuario se encuentra autenticado en el sistema con el nivel de acceso, acuerdo a su perfil.</td></tr>
</table>

Tabla 1. Descripción CU Inicio de sesión.

Tomar comanda	
Actores: Usuario (iniciador)	
Descripción: El caso de uso permite al usuario añadir comanda.	
Precondiciones: El usuario se encuentra autenticado en el sistema.	
Secuencia de eventos típica	
Actor	Sistema
	1. Sistema muestra pantalla con 4 botones.
2. Usuario hace click en el botón "Tomar comanda"	
	3. Sistema carga mesas y muestra lista de números de mesa.
4. Usuario elige el número de mesa.	
	5. Sistema crea un registro en la base de datos para la comanda. Muestra pantalla con todos los productos por familia.
6. Usuario va seleccionando cantidad de productos y cuando finaliza, da al botón de confirmar.	
	7. Muestra a modo de resumen, la comanda realizada.
8. Usuario Confirma el resumen de la comanda.	
	9. Guarda en base de datos la comanda y vuelve a mostrar la pantalla con los cuatro botones

Secuencia alternativa	
6. En cualquier momento, puede darle al botón de atrás y no se guardaría en base de datos ningún producto para esa comanda. (Volviendo al 4) 8. Puede darle al botón de atrás, volviendo a añadir o quitar productos de la comanda. (Volviendo al 6)	4. Si es una mesa ya ocupada, el sistema carga los registros de la comanda.
Postcondiciones: Estado mesa seleccionada queda como abierta.	

Tabla 2. **Descripción CU Tomar comanda.**

Traspasar mesa	
Actores: Usuario (iniciador)	
Descripción: El caso de uso permite al usuario cambiar una comanda de mesa.	
Precondiciones: El usuario se encuentra autenticado en el sistema.	
Secuencia de eventos típica	
Actor	Sistema
1. Usuario hace click en el botón "Traspasar mesa"	
	2. Sistema carga desde la base de datos, todas las mesas.
3. Usuario introduce dos números de mesa: el de origen y a cuál quiere pasarla y pulsa botón Aceptar.	
	4. Sistema comprueba que los números de mesa sean correctos.
5. Usuario confirma el traspaso de mesa.	
	6. Copia los registros de productos que hay en la mesa origen a la de destino. Vuelve a mostrar la pantalla con los cuatro botones
Secuencia alternativa	
	4. Si los dos números son el mismo, la mesa destino no es una mesa libre, la mesa origen no es una mesa ocupada, o no existen, muestra un mensaje de error por pantalla.
5. Usuario puede pulsar el botón de atrás, sin realizarse el traspaso de la mesa. (Volviendo al 3)	
Postcondiciones: -	

Tabla 3. **Descripción CU Traspasar mesa.**

<table>
<tr><th colspan="2">Unir mesas</th></tr>
<tr><td colspan="2">Actores:
Usuario (iniciador)</td></tr>
<tr><td colspan="2">Descripción:
El caso de uso permite al usuario seleccionar dos mesas y unirlas en una.</td></tr>
<tr><td colspan="2">Precondiciones:
El usuario se encuentra autenticado en el sistema.</td></tr>
<tr><td colspan="2">Secuencia de eventos típica</td></tr>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td>1. Usuario hace click en el botón “Unir mesas”

3. Usuario introduce dos números de mesa: el de origen y a cuál quiere pasarla y pulsa botón Aceptar.

5. Usuario confirma el traspaso de mesa.</td><td>2. Sistema carga desde la base de datos, todas las mesas con estado ocupado.

4. Sistema comprueba que los números de mesa sean correctos.

6. Copia los registros de productos que hay en la mesa origen a la de destino. Vuelve a mostrar la pantalla con los cuatro botones.</td></tr>
<tr><td colspan="2">Secuencia alternativa</td></tr>
<tr><td>5. Usuario puede pulsar el botón de atrás, sin realizarse la unión de las mesas.
(Volviendo al 3)</td><td>4. Si los dos números son el mismo, la mesa origen o destino no son mesas ocupadas, o no existen, muestra un mensaje de error por pantalla.</td></tr>
<tr><td colspan="2">Postcondiciones:
-</td></tr>
</table>

Tabla 4. **Descripción CU Unir mesas.**

<table>
<tr><th colspan="2">Pagar mesa</th></tr>
<tr><td colspan="2">Actores:
Usuario (iniciador)</td></tr>
<tr><td colspan="2">Descripción:
El caso de uso permite al usuario pagar la comanda de la mesa.</td></tr>
<tr><td colspan="2">Precondiciones:
El usuario se encuentra autenticado en el sistema.</td></tr>
<tr><td colspan="2">Secuencia de eventos típica</td></tr>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td>1. Usuario hace click en el botón “Pagar”</td><td></td></tr>
<tr><td></td><td>2. Sistema cargan las mesas con estado ocupado y muestra una lista con los números.</td></tr>
<tr><td>3. Usuario elige el número de mesa.</td><td></td></tr>
<tr><td></td><td>4. Sistema carga los productos, para esa comanda y los muestra a modo de resumen.</td></tr>
<tr><td>5. Usuario pulsa el botón de Pagar.</td><td></td></tr>
<tr><td></td><td>6. Sistema muestra una ventana emergente para seleccionar el método de pago: efectivo o tarjeta y un botón de atrás.</td></tr>
<tr><td>7. Usuario selecciona uno de los dos métodos de pago.</td><td></td></tr>
<tr><td></td><td>8. Sistema actualiza la comanda de la mesa con el estado de pagado efectivo o pagado tarjeta (según lo seleccionado). Y libera la mesa.
Vuelve a mostrar la pantalla con los cuatro botones</td></tr>
<tr><td colspan="2">Secuencia alternativa</td></tr>
<tr><td>3. o 7. Usuario puede pulsar el botón de atrás, no se realiza el cobro de la mesa. (Volviendo al 3)</td><td></td></tr>
<tr><td colspan="2">Postcondiciones:
-</td></tr>
</table>

Tabla 5. **Descripción CU Cobrar mesa.**

5.2 Diseño

En este apartado, explicaremos cómo se han planteado las tres soluciones que comprenden el proyecto: CrossApp, que es la aplicación multiplataforma; WebAPI, la aplicación web y AdminWeb, proyecto web.

El desarrollo se hará con una arquitectura en N capas, normalmente incluye una capa de presentación (en el proyecto la aplicación multiplataforma o la web de administración), una capa de negocio, otra capa lógica (que estarían unificadas en la API) y una capa de datos (la base de datos SQL). Por lo tanto, las capas solo interactúan con las inmediatamente superiores e inferiores, de manera que se abstraen funcionalidades y también proporciona mayor seguridad al desacoplar componentes.

5.2.1 Modelo de clases

5.2.1.1 CrossApp

Por un lado, tenemos las páginas que tiene nuestra aplicación y por otro, los servicios donde estará la lógica que se hará para que se siga el flujo correcto, en función de la interacción del usuario.

Habrá ocho pantallas, en las cuales tendremos los siguientes métodos:

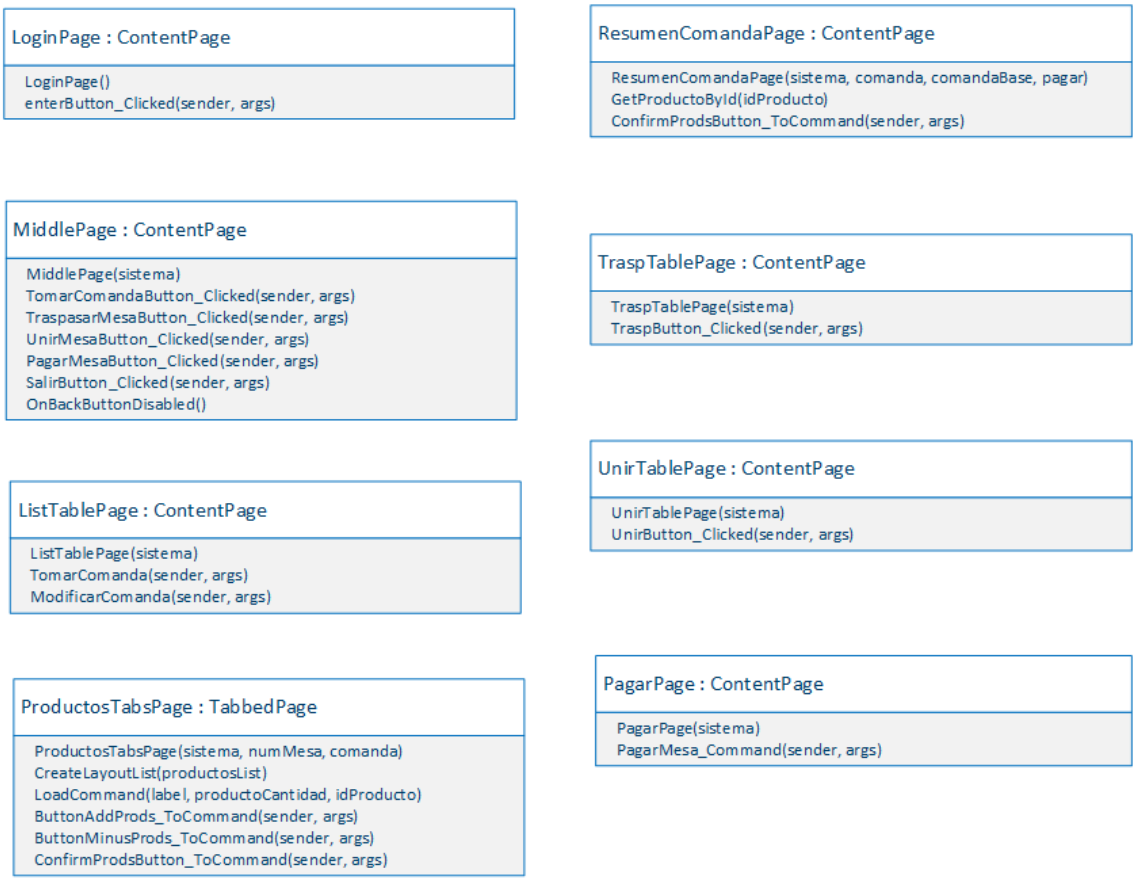

Figura 5. Modelo clases Vistas de CrossApp.

En primer lugar, tenemos la página de LoginPage para iniciar sesión. Después tenemos una página intermedia con los botones que redirigen a los métodos que realizan las acciones de la aplicación. El segundo botón de tomar comanda mostrará las páginas con las listas de las mesas (ListTablePage), que ejecutará los métodos de TomarComanda() o ModificarComanda() según corresponda, y las pestañas con los productos (ProductosTabsPage). Esta tiene los métodos para mostrar las listas de productos en las pestañas, cargar una comanda y sumar o restar productos a la cantidad que se muestra.

A continuación, ResumenComandaPage tiene un método para buscar los productos según su identificador para cargar la comanda generada.

Luego las páginas para traspasar o unir mesas (Trasp/UnirTablePage respectivamente), la página para pagar la mesa (PagarPage) y por último el botón de Salir.

Y los servicios son cuatro, para definir la gestión de usuarios, productos, mesas y comandas, y su comunicación con la WebApp.

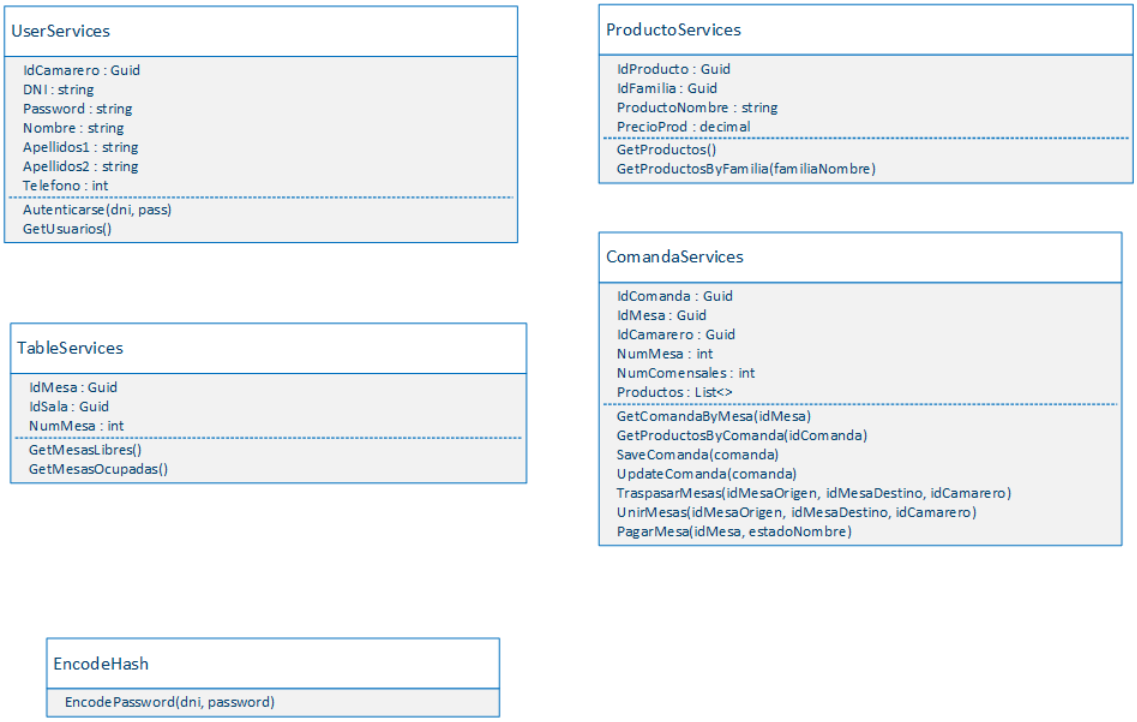

Figura 6. **Modelo clases Servicios de CrossApp.**

El método del servicio de usuario es para el inicio de sesión, para el que usaremos el método de la clase de EncondeHash para codificar la contraseña.

En las tablas haremos la sincronización de las mesas y usando dos métodos las diferenciaremos. Listando las mesas que tienen comandas con estado abierto, obtendremos las mesas ocupadas y eliminando éstas de la lista de todas las mesas, tendremos las mesas libres.

El servicio de productos se utilizará para recuperar todos los productos y en función de la familia.

Por último, el servicio de comandas será el que recupere la comanda con determinado identificador de mesa, o los productos de un identificador de comanda. También tendrá los métodos para guardar o actualizar las comandas; traspasar, unir o pagar mesas.

5.2.1.2 Web Application

La Web Aplication será a la que accederán las otras dos soluciones para tener acceso a la base de datos.

Mediante métodos GET (para extraer información de la base de datos) y POST (para modificar la información) gestionaremos los diferentes cambios que han de hacerse en la base de datos en todo el proceso activo de la aplicación o la web de administración.

Los controladores serán los que recibirán esas sentencias GET o POST, y llamarán a los métodos de los servicios correspondientes para ejecutar la acción pedida.

Por lo que, tenemos dos componentes principales: los controladores y los servicios (definidos en interfaces). Éstos últimos serán los que se comuniquen y hagan las peticiones a la base de datos.

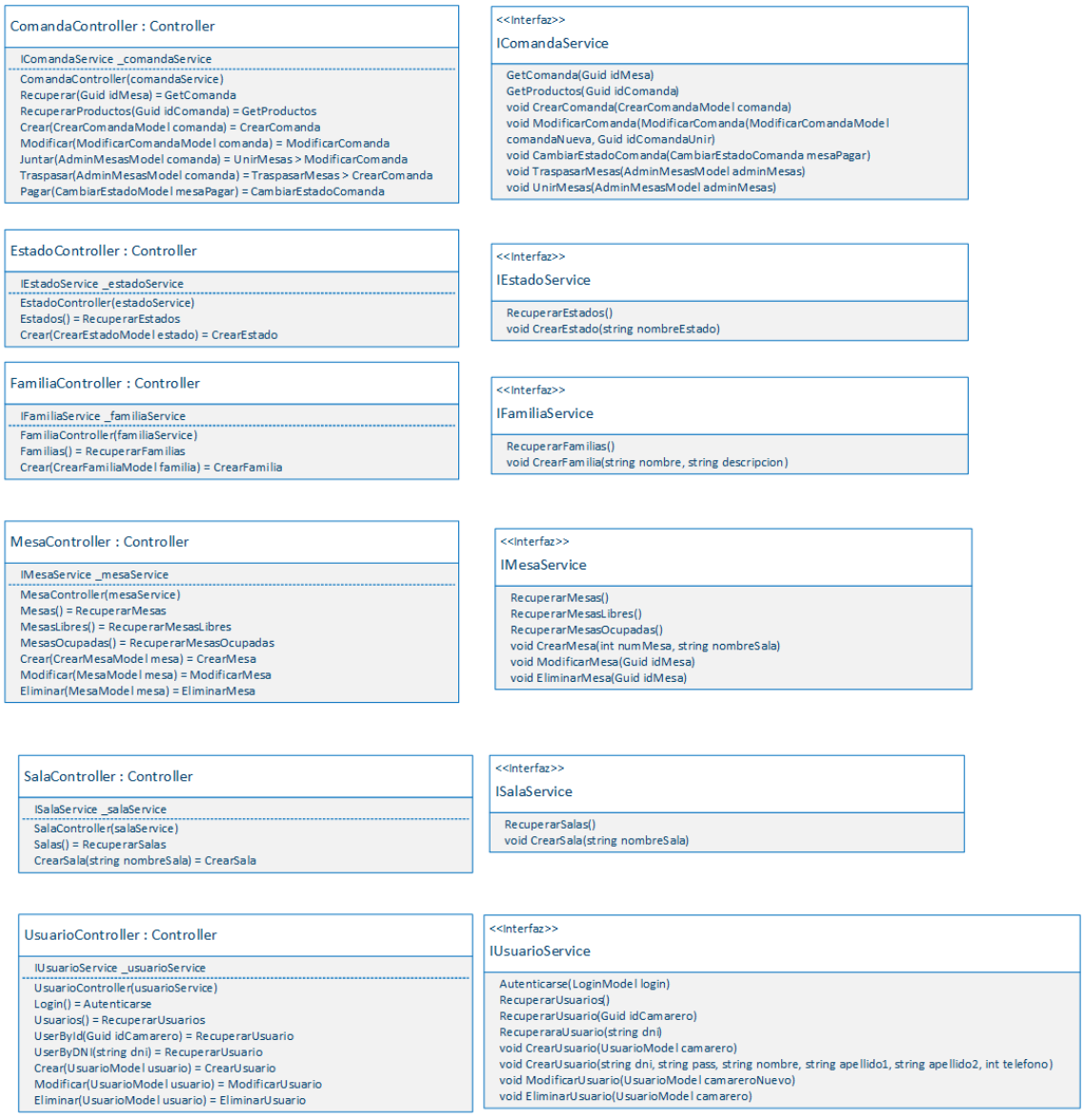

Figura 7. Modelo clases Controladores e Interfaces de servicios de CrossApp.

Podemos ver en los controladores, a la parte izquierda del igual, el método que corresponden con la llamada desde CrossApp o WebApp, y en la parte derecha de la igualdad tenemos los métodos de los servicios al que corresponden, que está implementado en el Servicio como indica el acuerdo de interfaz.

Las peticiones realizadas desde CrossApp a los controladores, se describirán más en detalle en los diagramas de secuencia.

5.2.1.3 AdminWeb

Por último, la web de administración está dividida en los tres componentes anteriormente mencionados: modelos, vistas y controladores.

Los modelos se corresponden con los que hay en la Web API, ya que, deben seguir el mismo patrón que la base de datos.

Vistas y controladores, tendremos tantos como páginas tiene la web. En este caso, la página de inicio (Home), otras tres para usuarios, mesas y productos (que son los tres elementos que he elegido para poder administrar mediante esta web). Y en cada clase habrá métodos de listar, crear, editar, ver detalles y eliminar.

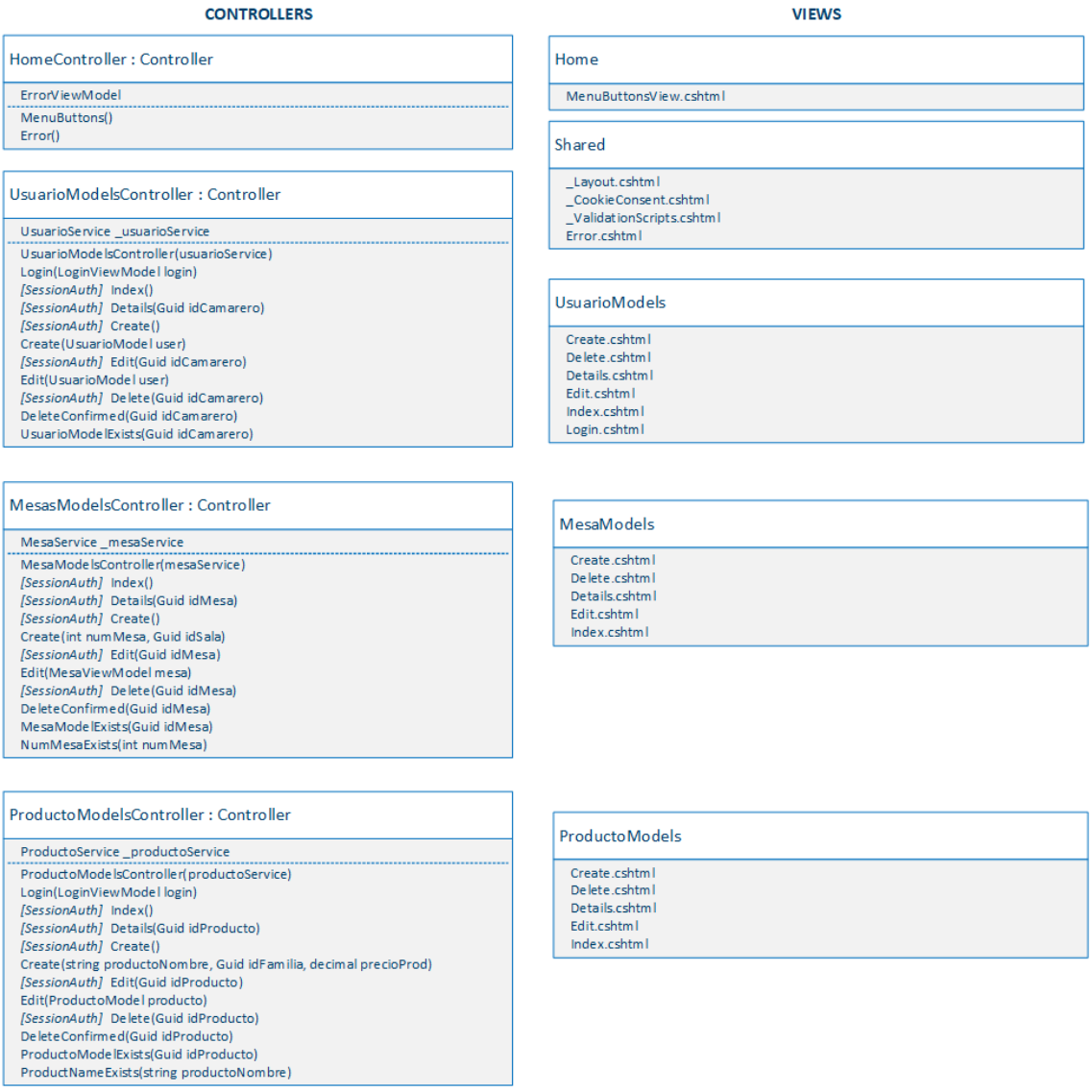

Figura 8. Modelo clases Controladores y Vistas de AdminWeb.

En el caso de usuarios, además tenemos el inicio de sesión y usaremos otra clase de Utilidades para mantener la sesión a través de cada página. Por lo tanto, si no se ha iniciado sesión no puedes navegar a través de las pestañas de la web.

Figura 9. **Modelo clases Controladores y Vistas de AdminWeb.**

5.2.2 Diseño base de datos

La base de datos es compartida por todo el proyecto, pero solo accede a ella WebApp. Consiste en ocho tablas, donde tendremos guardados los usuarios, productos y mesas.

Por un lado, tendremos los usuarios, donde almacenaremos los usuarios que van a usar la aplicación móvil y la web de administración. En esta tabla guardaremos su información y credenciales. También, por simplicidad, se añadieron a esta tabla dos campos flag para diferenciar los tipos de accesos a la web. El administrador será el que tenga el campo "OwnerWeb" activo, y por tanto, será el que tenga permiso de escritura de Usuarios, en la web podrá crear, editar o eliminar usuarios, productos y mesas. El resto de usuarios del sistema podrán acceder a la web con permisos de lectura sobre usuarios, y de escritura sobre productos y mesas, si tienen el campo "EnableWeb" activo. Si tienen este campo inactivo, accederán a la aplicación móvil pero no tendrán acceso a la web de administración.

Por otra parte, tenemos las mesas estarán agrupadas en salas y los productos que están agrupados en familias (éstas diferenciarán las pestañas de nuestra aplicación).

A su vez, tendremos las comandas donde estarán los datos que relacionarán, el camarero, la mesa y el estado de la misma (abierto, pagado, traspasado…), además del número de comensales.

Por último, la tabla intermedia Comanda_Producto, que será la que relacione la comanda con cada producto que se pide, su cantidad, y el precio calculado (con la cantidad y el precio unitario).

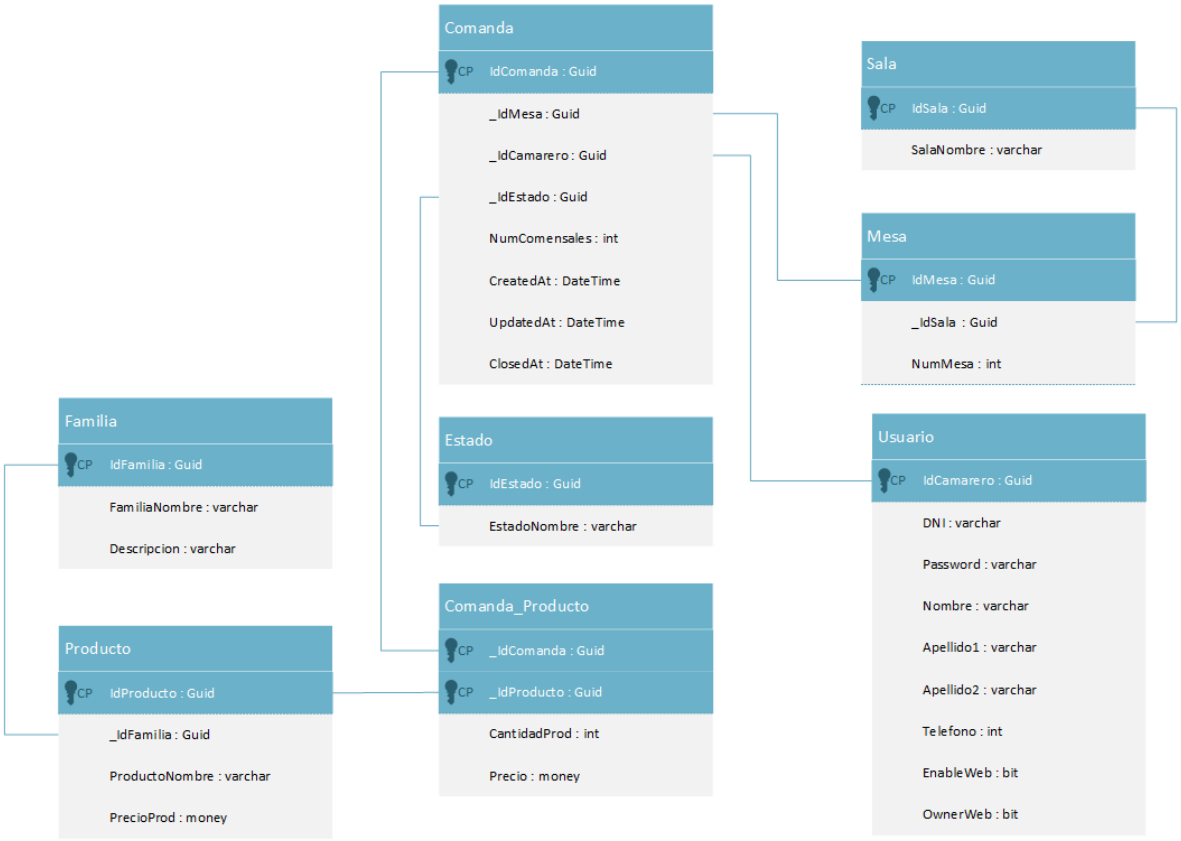

Figura 10. **Modelo Lógico base de datos.**

5.2.3 Diagramas de secuencia

Una vez definidas las clases que existirán en el proyecto, las operaciones que desarrolla cada una y conociendo las interacciones posibles entre los usuarios y la aplicación, describiremos en detalle los diagramas de secuencia correspondientes a cada caso de uso, de la aplicación multiplataforma.

5.2.3.1 Inicio de sesión

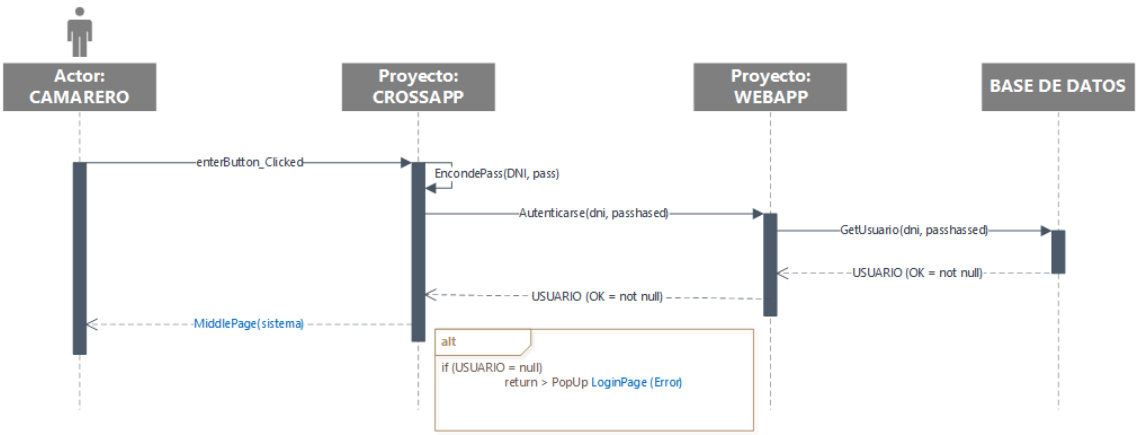

Figura 11. **Diagrama secuencia del Inicio de sesión.**

Durante el inicio de sesión a nuestra aplicación, cuando el actor pulse el botón Entrar, el proyecto de CrossApp generará un Hash de la contraseña introducida junto con el DNI (con la librería de criptografía SHA256 que nos ofrece Visual Studio). Mediante este algoritmo matemático de encriptación obtenemos, para una misma entrada, una serie de caracteres alfanuméricos con longitud fija. Y este valor hash es el que se guardará en la base de datos, para mayor seguridad del usuario, ya que no viajan, ni son guardadas las contraseñas en plano.

Con la información del DNI introducida por el usuario y el valor del hash, se hará una petición a la WebApp de autenticación. Ésta mediante el método "GetUsuario" hará la consulta de búsqueda del usuario en la base de datos, que coincidan los valores anteriores.

En caso de encontrarlo, devuelve toda la información de este en un objeto usuario, que recibirá la WebApp, interpretará y traspasará a CrossApp que dará acceso a la siguiente página de la aplicación que es "MiddlePage" junto con la información del sistema, es decir, los datos del usuario autenticado.

En el caso que el usuario correspondiente al DNI y el hash de la contraseña, no exista en base de datos, devolverá un nulo. WebApp recibe el objeto Usuario como nulo y lo traspasa a CrossApp, que, mostrará un mensaje de error en la página de inicio de sesión de la aplicación.

5.2.3.2 Tomar comanda

Al pulsar el primer botón desde la página "MiddlePage", Tomar Comanda, la solución hace una petición GET a la WebApp para recuperar todas las mesas, ésta a su vez, hace dos consultas a la base de datos: RecuperarMesasOcupadas(), devuelve una lista de las mesas que tengan una comanda con estado abierto, mesas ocupadas; y RecuperarMesasLibres() devuelve una lista de las mesas eliminado del total de mesas las ocupadas. Ambos listados se muestran la siguiente página "ListTablePage".

Según de qué lista se seleccione el número de mesa (libres u ocupadas), se diferenciarán dos procesos distintos: Nueva o Modificar comanda, respectivamente.

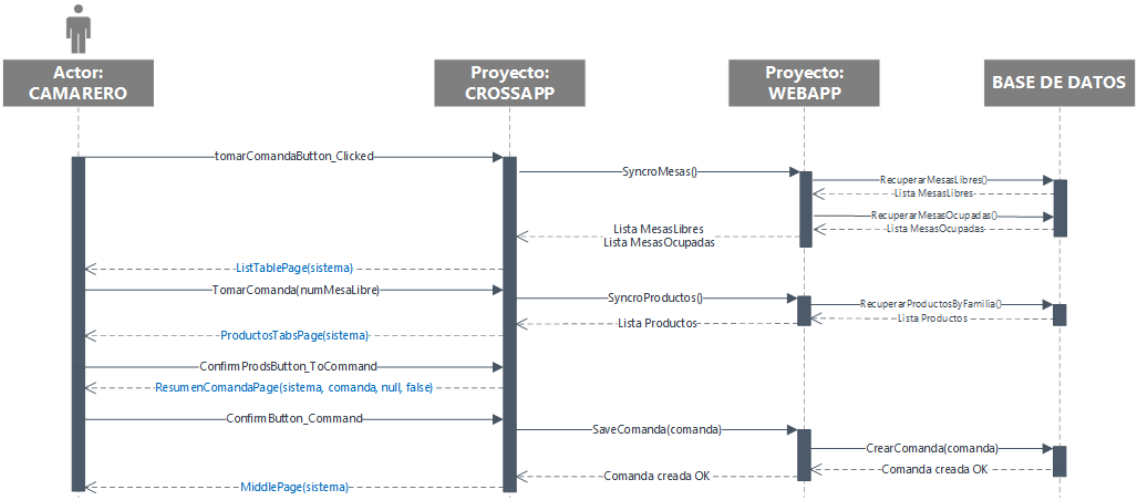

Figura 12. Diagrama de secuencia Nueva comanda.

En el primero de los casos, cuando se pulsa el número de una mesa libre, se ejecuta el método "TomarComanda", que en primer lugar hará una petición a la WebApp para recuperar de la base de datos todos los productos por familias. Y seguidamente muestra la página con

pestañas “ProductosTabsPage”, donde cada una será una familia y en ella habrá una lista con sus productos.

Cuando desde esa página se pulsa el botón de confirmación, se pasa a la página de “ResumenComanda” con los parámetros de: comanda actual y el bit de pagar como false. Se cargará dicha comanda tomada y se mostrarán por pantalla la lista de los productos, cantidad, precio por producto y precio total de la comanda.

Y si se confirma este resumen de comanda, se hará una petición a la WebApp que ejecutará el método para guardar la comanda y crear en la base de datos los registros en la tabla Comanda (con el estado Abierto) y Comanda_Producto, con la lista de productos seleccionados.

Por último, cuando CrossApp reciba la respuesta mostrará la página inicial MiddlePage.

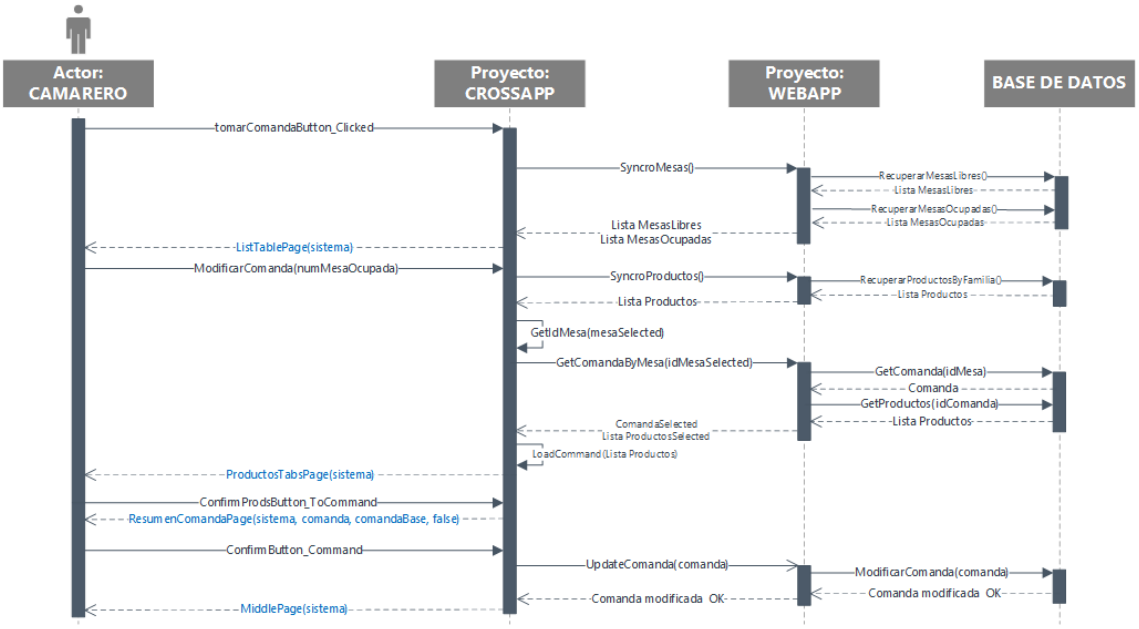

Figura 13. **Diagrama de secuencia Modificar comanda.**

Si el actor, en este caso el camarero, selecciona una mesa que está ocupada de la página “ListTablePage”, se ejecuta el método ModificarComanda, primero haremos una petición a la WebApp y ésta una consulta a la base de datos para recuperar todos los productos por familias. Después consultaremos del listado de mesas recuperado al inicio, el identificador de esa mesa, para hacer una consulta a la base de datos desde la WebApp y obtener la comanda con estado “Abierto” para ese identificador de mesa, y sus productos.

Así, cuando CrossApp tiene esa información carga las cantidades de los productos recuperados en la lista de productos que se sincronizó al inicio y así en la página de ProductosTabsPage, se mostrará la cantidad actualizada de los productos que ya hay en esa comanda.

Cuando confirmamos la actualización de esa comanda, CrossApp le pasa como parámetros al método de ResumenComanda, además de la comanda actual (solo con los productos que se han modificado), la comanda base (la que habíamos recuperado previamente de base de datos) y el bit de pagar como false. Por lo que, a la hora de mostrar el listado de productos, pondrá primero los de la comanda base, luego los que estaban en la comanda base pero se ha actualizado la cantidad y finalmente los nuevos.

Y para finalizar el proceso, cuando se confirma en “ResumenComanda”, CrossApp deberá actualizar la comanda ya abierta. Por lo que se enviará a la WebApp una petición con la lista

actualizada de productos y el identificador de la comanda, para cambiar los registros con los nuevos datos en la tabla Comanda_Producto para ese identificador de comanda en concreto.

Cuando CrossApp reciba la respuesta de la WebApp, redirigirá a la página MiddlePage.

5.2.3.3 Traspasar mesa

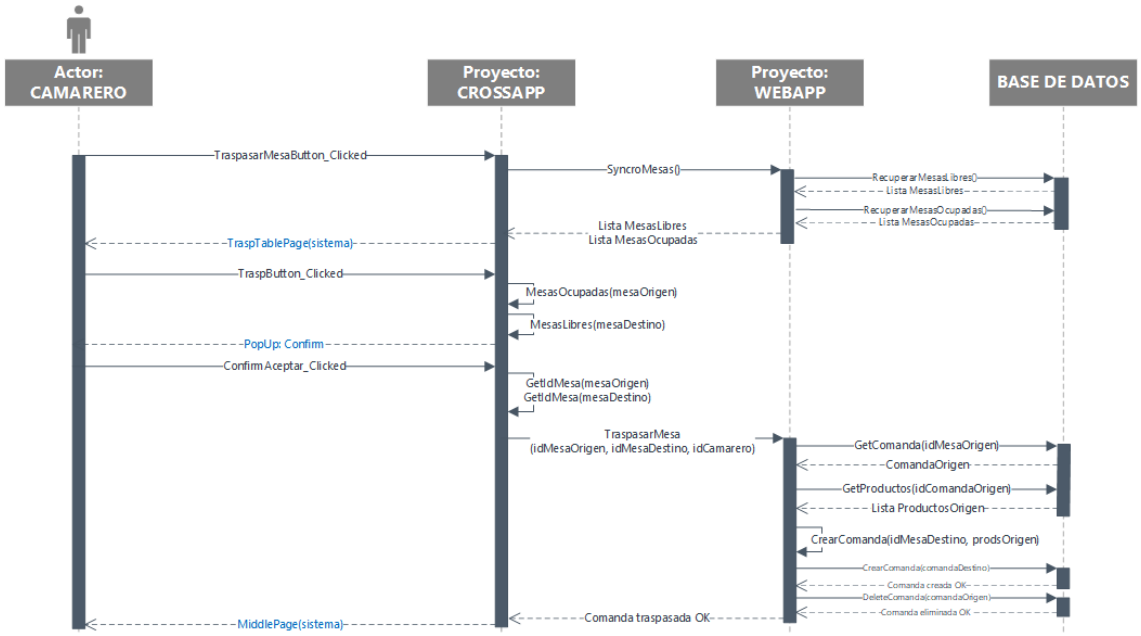

Figura 14. Diagrama de secuencia Traspasar mesa.

Pulsando el botón de “Traspasar Mesa” desde la página “MiddlePage”, se hace una petición GET a la WebApp de las mesas y ésta hace la consulta para recuperar las listas de las mesas en la base de datos.

La solución CrossApp muestra la página de “TraspTablePage” y una vez introducidos los números de mesa origen y destino, tras pulsar el botón de Aceptar consultará si las mesas introducidas están dentro de los listados de mesas ocupadas o libres, respectivamente. Si se han introducido los números de mesa correctamente, aparecerá una ventana emergente de confirmación del cambio. Y al pulsar Aceptar, buscará el identificador de esas dos mesas dentro del listado de las mesas recuperado al inicio de la ejecución.

CrossApp guardará los cambios enviando la información con una petición a la WebApp, que hará una consulta a la base de datos para obtener el identificador de la comanda para la mesa origen con estado Abierto y así recuperará la lista de productos de la comanda de origen.

A continuación, creará un nuevo objeto comanda con el identificador de la mesa destino y la lista de productos recuperados de la mesa origen, y ejecutará el método para crear en la base de datos la nueva comanda. Y posteriormente, eliminaremos la comanda origen.

Al recibir la respuesta de la API, CrossApp redirigirá a la página MiddlePage.

5.2.3.4 Unir mesa

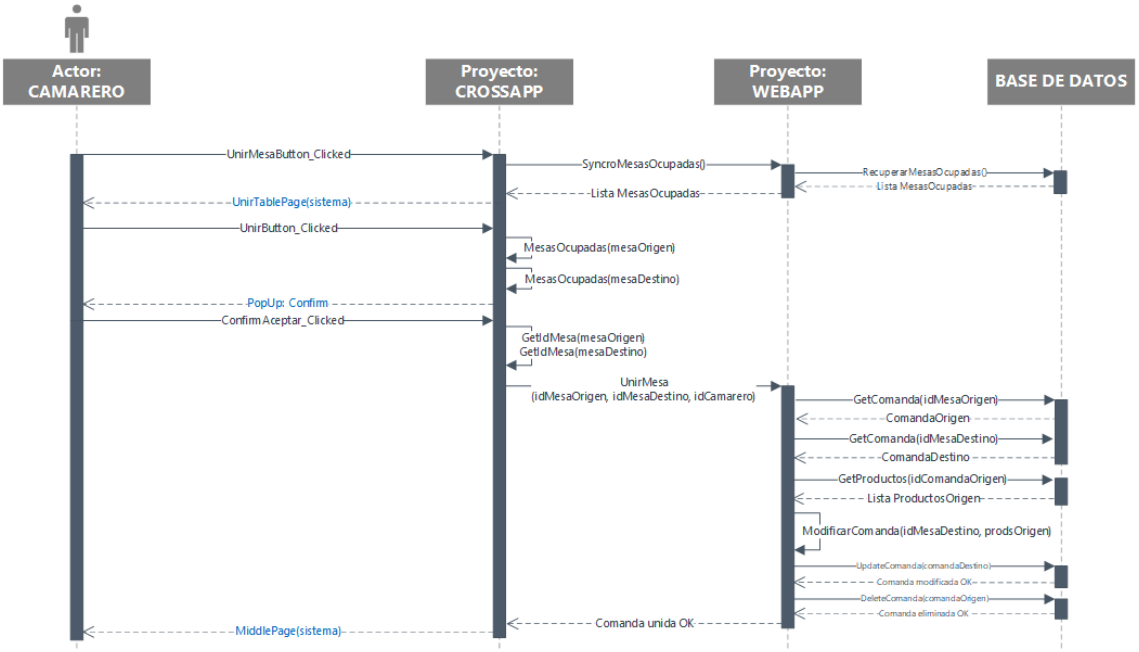

Figura 15. Diagrama de secuencia Unir mesa.

Si desde la página de "MiddlePage" pulsamos el botón de "Unir Mesa", se hace una petición para sincronizar las mesas ocupadas, haciendo la consulta desde la WebbApp a la base de datos para recuperar el listado.

CrossApp muestra la página de "UnirTablePage" y una vez introducidos los números de mesa origen y destino, se comprueba que ambos son mesas ocupadas. Siendo así, se mostrará una ventana emergente para confirmar el cambio y una vez aceptado, buscará el identificador correspondiente de los números de mesas introducidos, en la lista de mesas recuperada al inicio de la ejecución.

CrossApp lanza una petición asíncrona a la WebApp que hará una consulta a la base de datos para recuperar la comanda abierta para cada mesa y luego otra, obteniendo el listado de productos de la mesa origen. Después creará un objeto comanda con el identificador de la mesa destino y los productos de la mesa origen, para ejecutar la actualización de la comanda para la mesa destino. Y posteriormente, eliminaremos la comanda origen.

Y volverá a mostrar la página "MiddlePage", cuando obtenga la respuesta de la API.

5.2.3.5 Pagar mesa

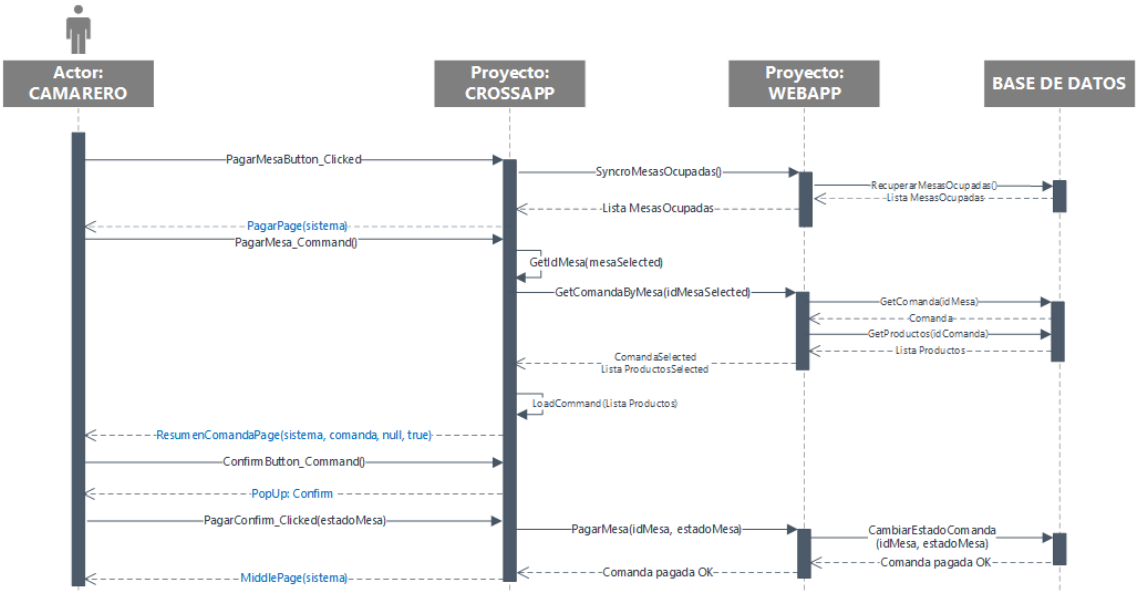

Figura 16. Diagrama de secuencia Pagar mesa.

Finalmente, si el actor pulsa el último de los botones de "MiddlePage", se ejecutará la sincronización de las mesas ocupadas, que nos devuelve la WebApp al hacer la consulta a la base de datos de las que tienen una comanda abierta. Se mostrará la página "PagarPage" con el listado de mesas ocupadas (si hay) y al pulsar uno de los números de mesa, buscaremos el identificador para ese número de mesa en la lista recuperada al inicio.

Mediante la WebApp haremos la consulta a la base de datos para obtener la comanda, y con su identificador la lista de productos para esa comanda y los mostraremos en la pantalla "ResumenComanda" con parámetro de pagar activo, por lo que, en esa vista el botón superior de confirmar ahora será de Pagar.

Cuando se pulsa el botón de Pagar, se mostrará una ventana emergente para seleccionar el método de pago. Seguidamente enviará la petición a la WebApp de PagarMesa con el identificador de la mesa y el estado seleccionado. Ésta ejecutará una consulta a la base de datos para cambiar el estado del registro de la tabla de Comanda para ese identificador de mesa.

Mostrará la página MiddlePage, en cuanto reciba la respuesta de la WebApp.

5.2.4 Flujogramas del funcionamiento

A continuación, se detallarán los flujogramas de funcionamiento de cada uno de los casos de uso que se han definido anteriormente.

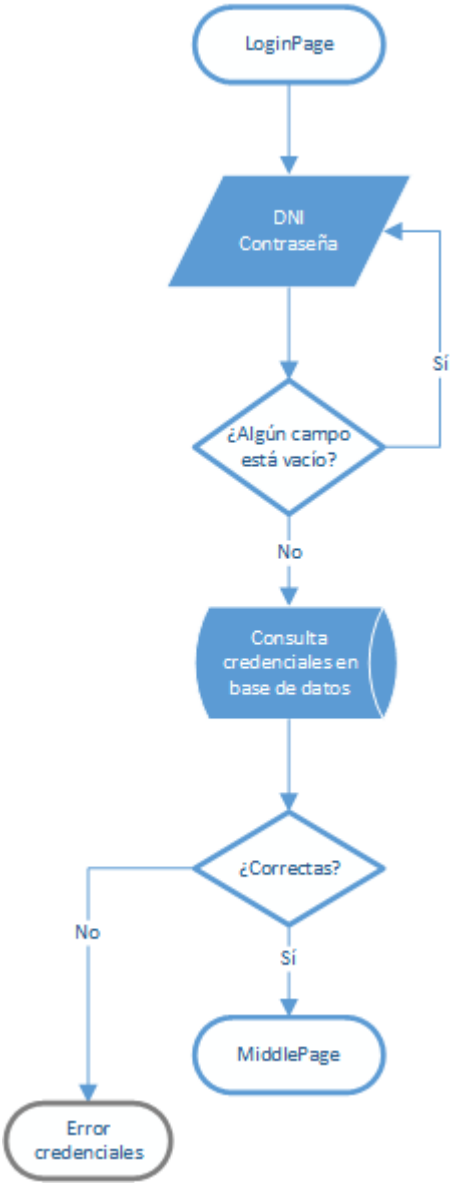

Figura 17. Flujograma Inicio de sesión.

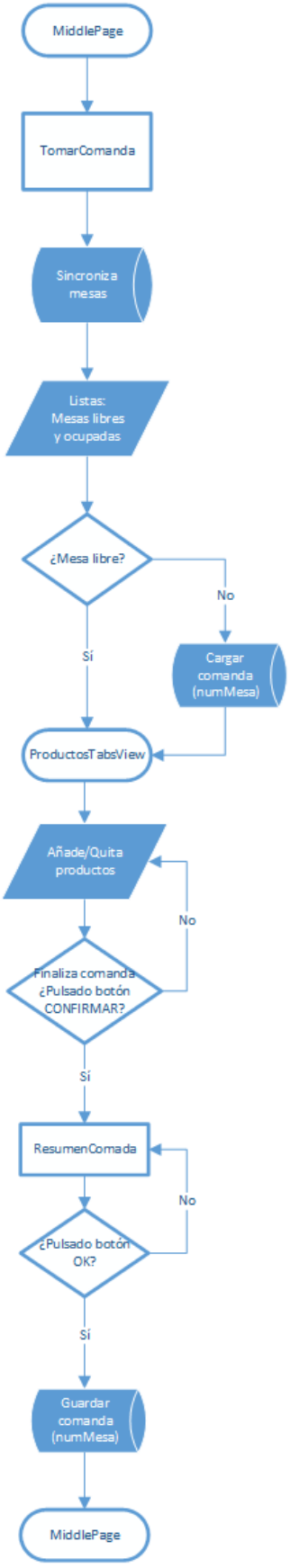

Figura 18. **Flujograma Tomar comanda.**

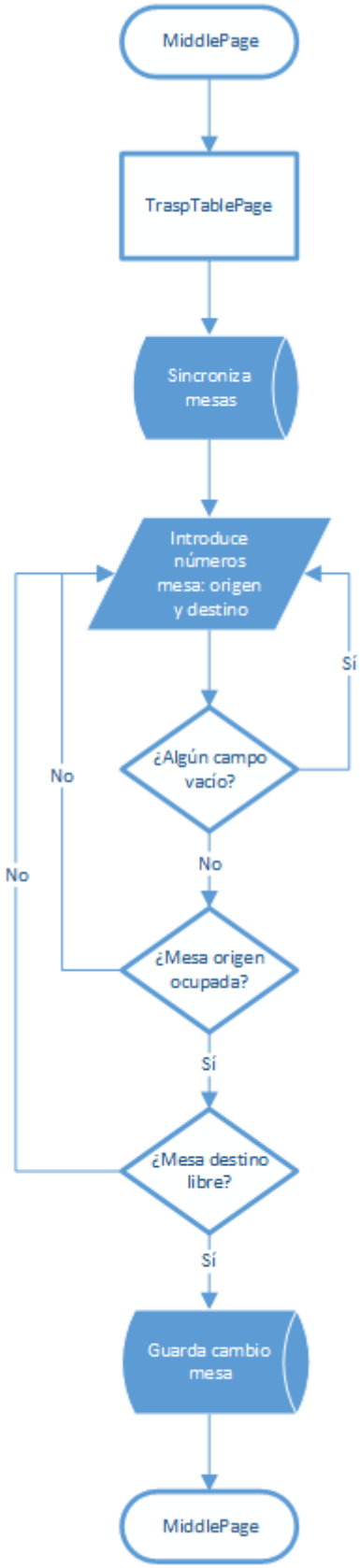

Figura 19. **Flujograma Traspasar mesa.**

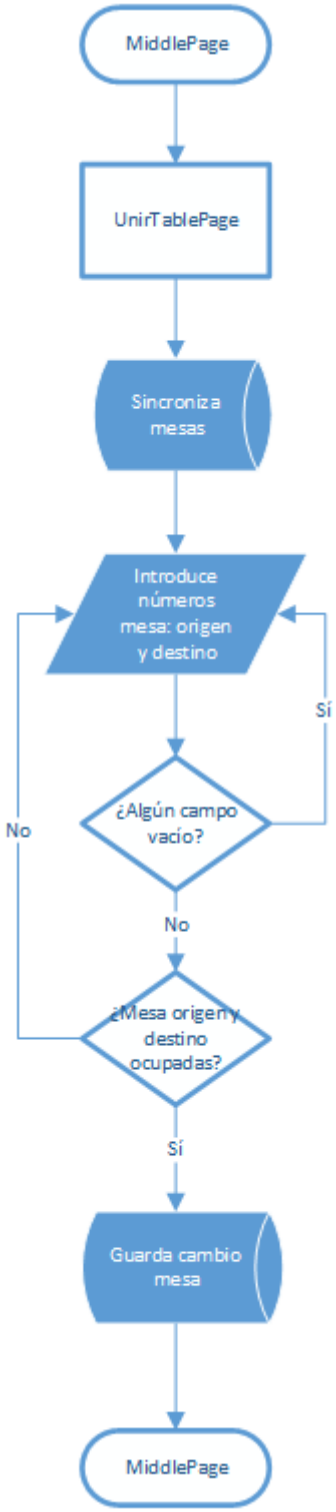

Figura 20. **Flujograma Unir mesa.**

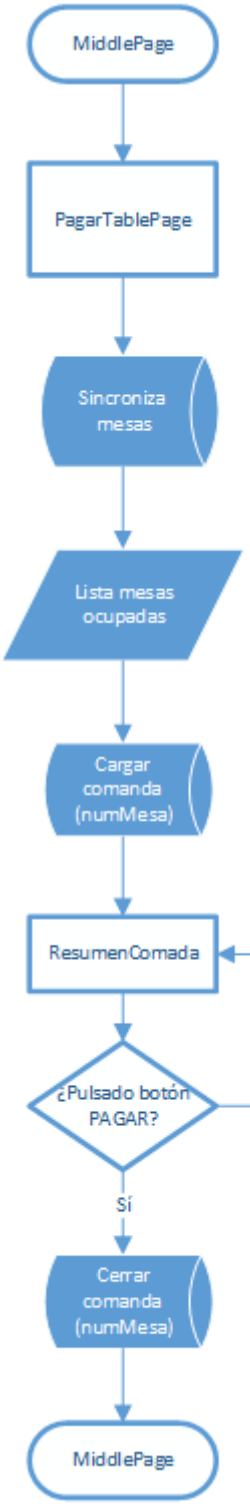

Figura 21. **Flujograma Pagar comanda.**

5.2.5 Diseño de interfaz de usuario

Al entrar en la aplicación la primera pantalla que se nos muestra es la de Inicio de sesión, donde podrá introducir sus credenciales en las dos entradas de texto y el botón de Entrar.

Figura 22. **Interfaz del inicio de sesión.**

Si las credenciales no son correctas, aparecerá una ventana emergente del error.

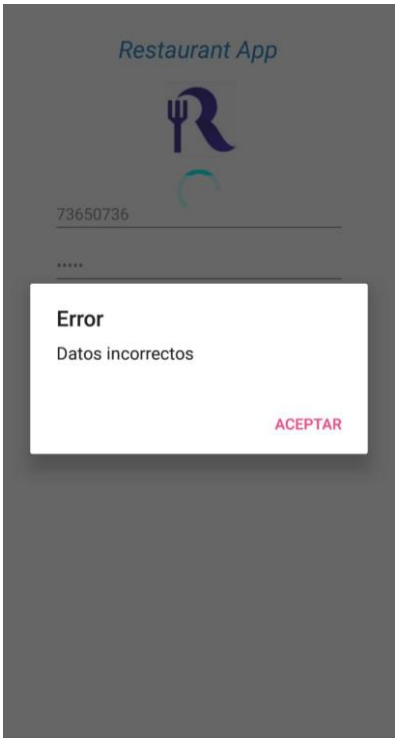

Figura 23. **Interfaz del error en inicio de sesión.**

Una vez ha introducidos correctamente los datos de inicio de sesión, se muestra una pantalla intermedia con el nombre del camarero autenticado, con cuatro botones (con las acciones a realizar) y un botón de salir para cerrar la aplicación.

Figura 24. **Interfaz pantalla principal.**

En caso de que se pulse el botón "Tomar comanda" aparecen dos listas de mesas: las libres y ocupadas (si hay), y cada lista puede hacer *scroll* pudiendo ver todos los números de mesa ordenados ascendentemente.

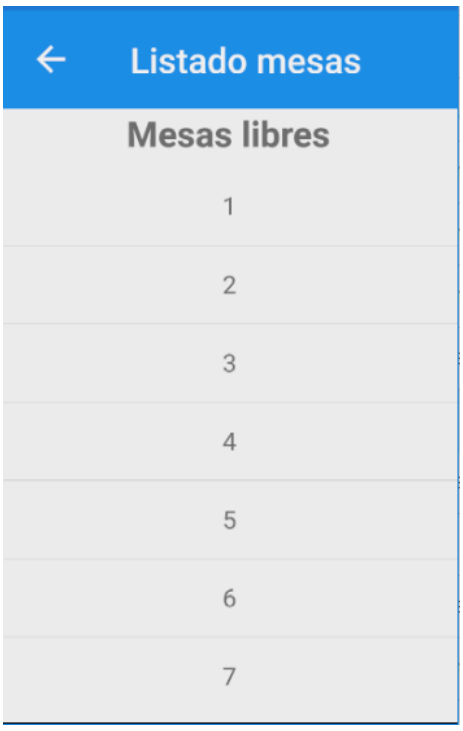

Figura 25. **Interfaz listado mesas libres.**

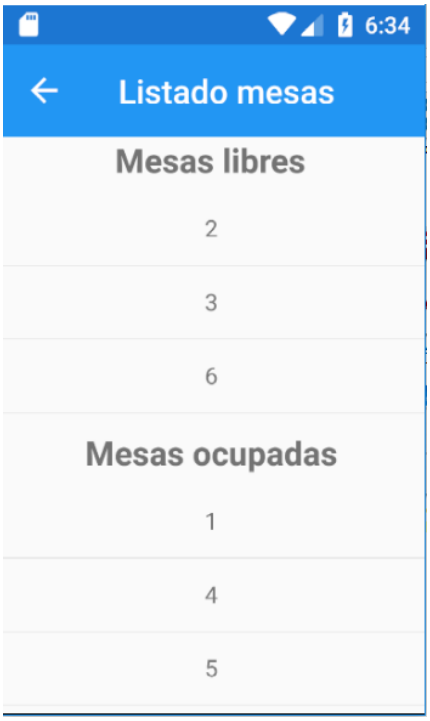

Figura 26. **Interfaz listados mesas.**

Pulsando cualquiera de los números se mostrará un menú de pestañas donde en la cabecera aparece el número de mesa seleccionado y cada pestaña es una familia de productos (en nuestro caso: bebidas, entrantes, carne, pescado y postre/café). Pulsando cualquiera de los iconos superiores o deslizando horizontalmente, cambiamos de pestaña y aparece una lista de los productos de esa familia con el precio unitario, unos botones para añadir o eliminar ese producto y la cantidad.

Cuando acabamos de navegar por las pestañas y añadir los productos, pulsaremos el botón de confirmar o pulsando la flecha del margen superior izquierdo, podemos volver atrás sin guardar ningún cambio para esa mesa.

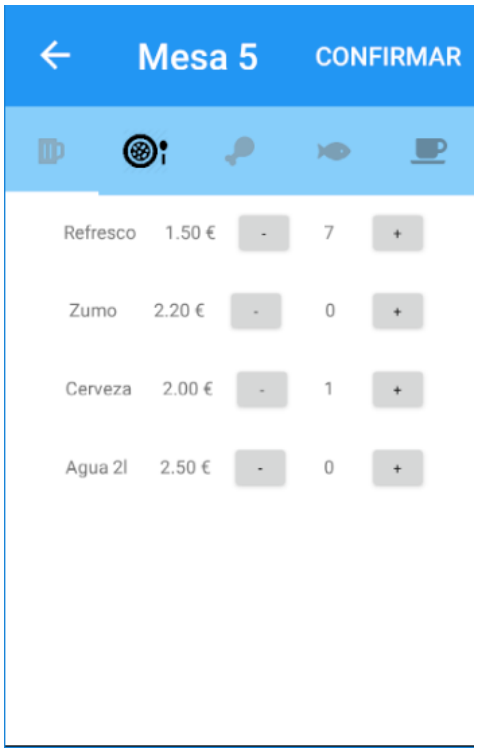

Figura 27. **Interfaz productos por familias.**

Al pulsar el botón de confirmar, veremos el resumen total de la comanda para esa mesa. Únicamente al pulsar OK se guardará nuestra comanda, tendremos la posibilidad de ir atrás, seleccionar los productos que falten y volver al resumen total.

Figura 28. **Interfaz resumen comanda.**

Una vez guardada esa comanda, volveríamos a la pantalla intermedia (Figura 24). Pulsando "Traspasar mesa" (Figura 29) o "Unir mesas" (Figura 32) obtenemos las pantallas con dos cajas de texto para introducir números y el botón de aceptar.

Siendo los números correctos, se nos muestra una ventana emergente de confirmación del cambio, con los números de mesa seleccionados y dos botones: Atrás y Aceptar (Figura 30 y 33). Y es pulsando el segundo de ellos, cuando guardaremos el cambio en base de datos. Si pulsamos Atrás, no se registraría ninguna modificación en las mesas introducidas.

Figura 29. **Interfaz traspasar mesas.**

Figura 30. **Interfaz confirmar traspaso mesa.**

Para traspasar la mesa, deberemos introducir como origen, una mesa que esté ocupada y como destino, una libre. En caso contrario, o ser el mismo número en las dos entradas de texto, aparecerá un mensaje de error como este (Figura 30).

Figura 31. **Interfaz error traspaso mesa.**

Y en el caso de querer unir mesas, deberían ser dos ocupadas. Por lo que, si el número no existe o alguna de ellas está libre, también nos aparecería un mensaje de error (Figura 33).

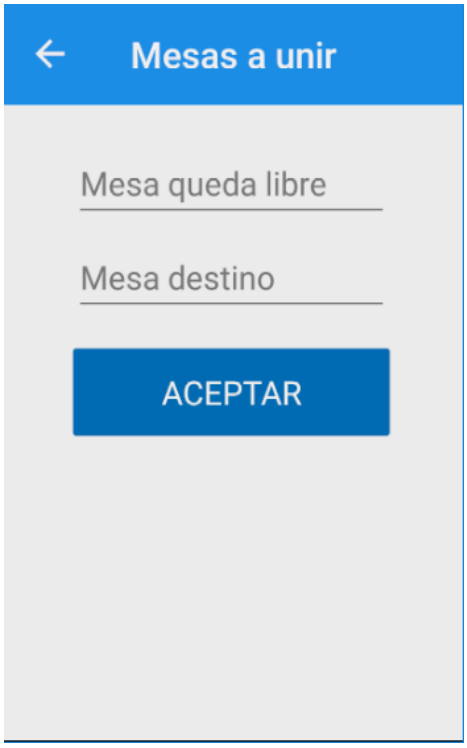

Figura 32. **Interfaz unir mesas.**

Figura 33. **Interfaz confirmar unir mesas.**

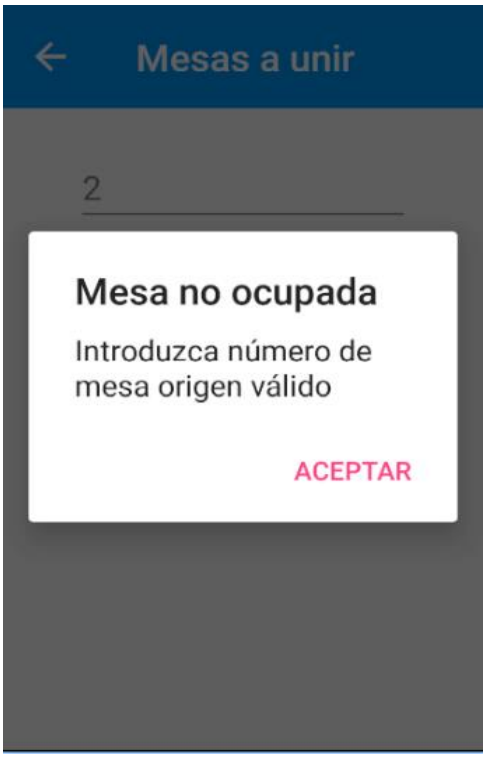

Figura 34. **Interfaz error unir mesa.**

Y, por último, pulsando el botón de "Pagar" de nuestra página intermedia (Figura 24), podríamos seleccionar un número de mesa ocupada (si hay), nos mostraría el resumen de la comanda.

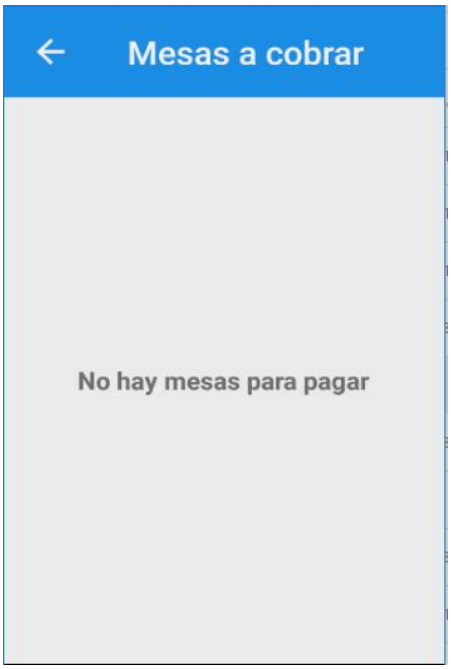

Figura 35. **Interfaz lista vacía de mesas pagar.**

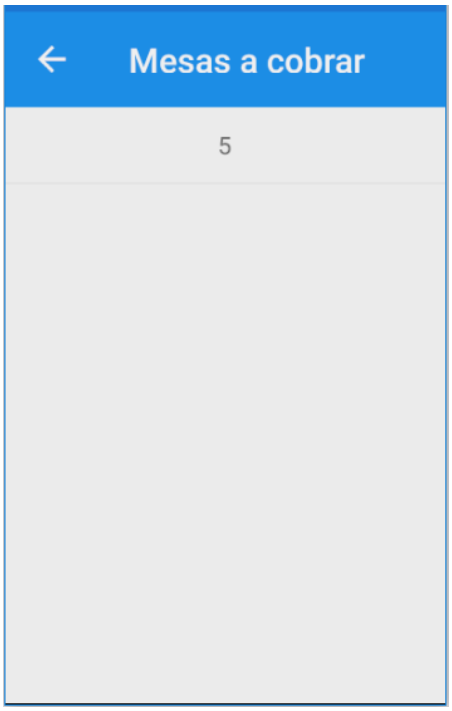

Figura 36. **Interfaz lista mesas ocupadas a pagar.**

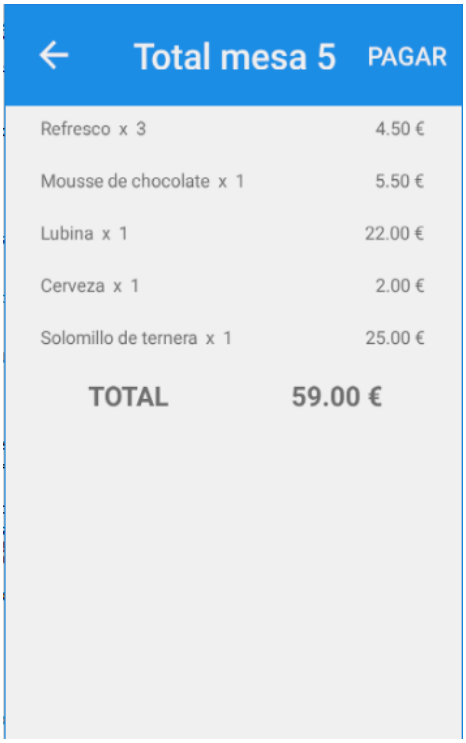

Figura 37. **Interfaz resumen mesa a pagar.**

Pulsando pagar, aparecería una ventana emergente para seleccionar el método de pago (en efectivo o tarjeta) y cerraremos la comanda.

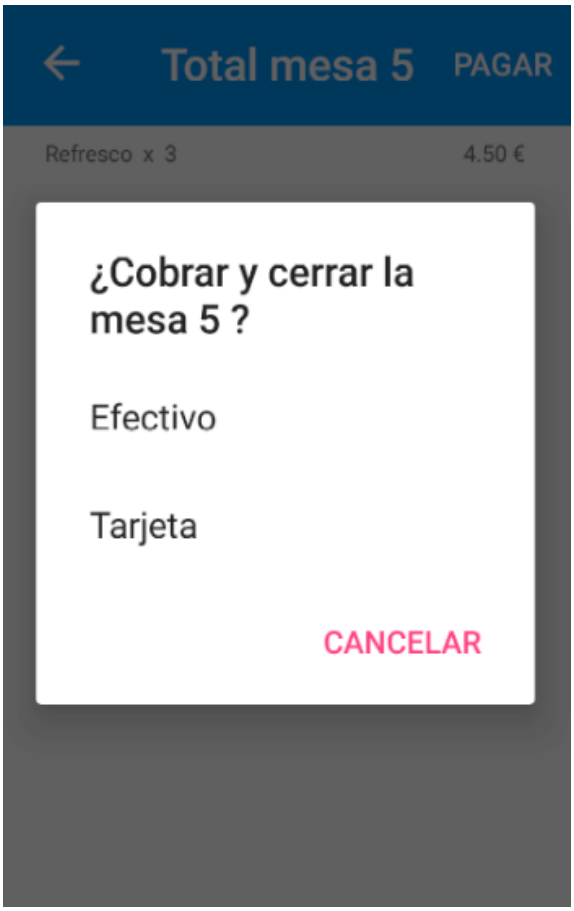

Figura 38. **Interfaz métodos pago.**

Capítulo 6. Desarrollo y resultados del trabajo

Durante este desarrollo se ha utilizado la plataforma Microsoft Azure para realizar una implantación PaaS (Platform as Service) [13], es decir la API y la web se desplegaron en el mismo AppService de Azure y la base de datos también se desplegó directamente en la nube. Esto se traduce en facilitar las tareas de gestión de la infraestructura, y en un desarrollo más rápido, simple y eficiente, además de poseer todas las ventajas de la nube, como la escalabilidad, la reducción de costes a largo plazo, y facilitación de la automatización de procesos y tareas, entre otras.

En conclusión, no fue necesario ningún trabajo de configuración en la infraestructura, ya sea de red, seguridad, o disponibilidad del sistema. A continuación, se muestra un esquema de cómo sería la infraestructura resultante.

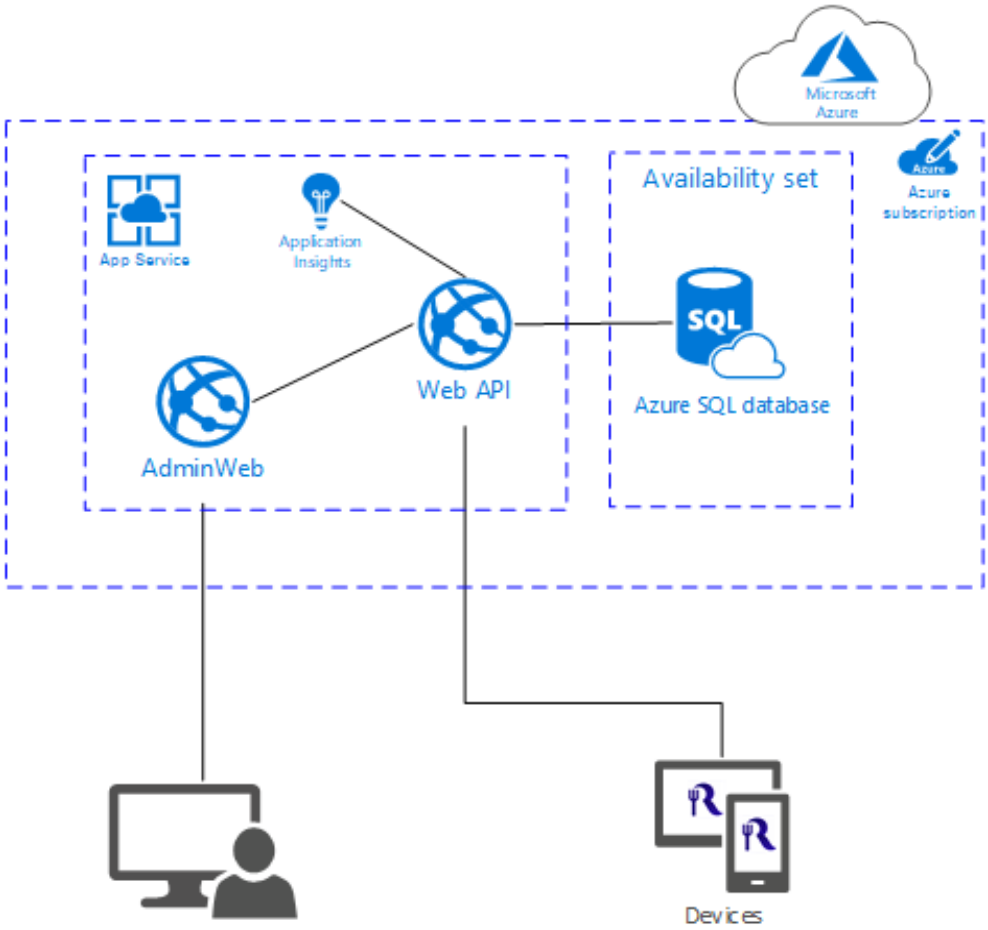

Figura 39. **Esquema infraestructura.**

En resumen, el flujo general del sistema consiste en una aplicación móvil - que se instalará en cada uno de los dispositivos móviles desde los cuales serán tomadas las comandas - y acciones que realizan los actores sobre la misma. Estas acciones lanzan peticiones POST a la capa de negocio, una WEB API con diversos servicios la cual está alojada en la nube. A su vez, esta WEB API ataca una base de datos alojada también en la nube, y esta información fluye de nuevo hacia el cliente, donde es presentada a los usuarios.

De la misma manera, la web de administración accedería a la capa de negocio de la WEB API. Los usuarios entrarían a la web y, al estar en el mismo plan de servicio que la WEB API, se comunica con ella y con la base de datos.

Igualmente, la WEB API también está conectada al Application Insights, un servicio de análisis extensible que supervisa la aplicación activa, incluso en producción se puede monitorizar para la detección y diagnóstico de problemas de rendimiento. También se podría utilizar para analizar patrones de uso de los usuarios. Además, esta herramienta puede estar integrada en todos los componentes de la solución de Azure, por lo que podríamos tener unificada toda la monitorización, pero conservando la claridad de las trazas [14].

6.1 Pruebas

Mediante Application Insights, podemos hacer un seguimiento de las rutas de acceso de uso para evaluar el éxito de cada servicio implementado [15]. Al entrar en el portal de Azure y seleccionando Application Insights, obtenemos un resumen de los componentes que están enlazados con esta herramienta (en mi caso, del App Service con TFGrofolal, solo está conectada la API), las llamadas que han hecho y el tiempo total que han tardado.

Figura 39. **Resumen llamadas API de Application Insights.**

Si seleccionamos la pestaña de Rendimiento y vemos los registros que se han generado en la herramienta, podemos seguir las trazas de las interacciones con la WebApp.

En la siguiente Figura podemos ver las llamadas a la API mediante peticiones GET y POST para que CrossApp obtuviera la información necesaria para su funcionamiento. Esta tabla nos da información de la petición realizada, las veces que se ha llamado durante estas pruebas y las duraciones de esos procesos de la API (media, percentil 50, percentil 95 y percentil 99).

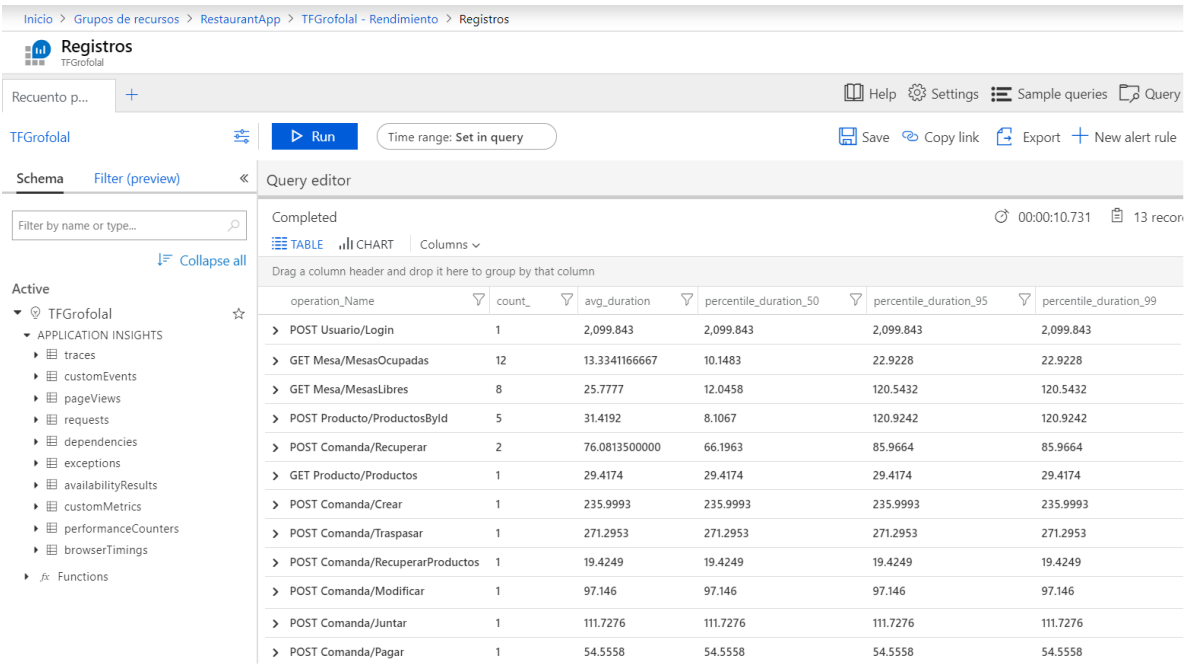

Figura 40. **Registros llamadas API de Application Insights.**

Podemos comprobar que todas las ejecuciones han funcionado correctamente, ya que, no obtenemos ningún fallo en la correspondiente gráfica:

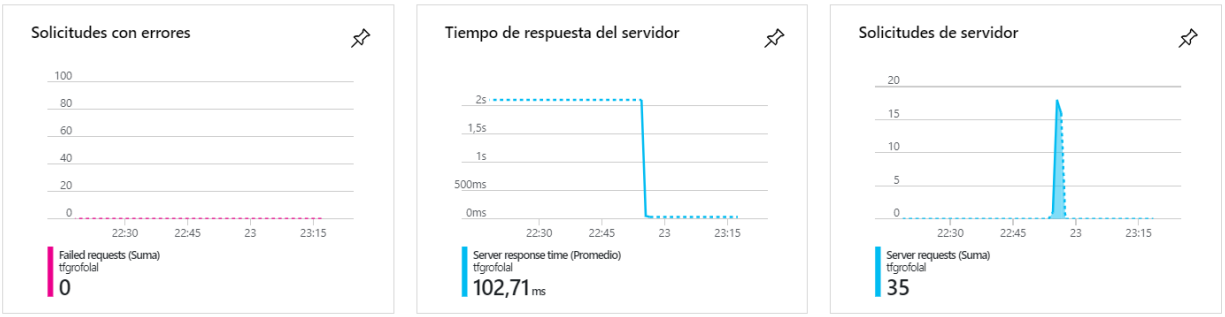

Figura 41. **Gráficas resultados llamadas API de Application Insights.**

En definitiva, además de las pruebas hechas en la aplicación en un dispositivo Android, se puede ver que se han testeado correctamente las funcionalidades que he implementado.

Capítulo 7. Conclusiones y propuesta de trabajo futuro

7.1 Objetivos alcanzados

A lo largo del trabajo se ha detallado cómo se ha hecho posible el desarrollo de una aplicación multiplataforma. En este sentido, la motivación de la aplicación exigía unos requisitos para asegurar su funcionalidad y eficacia en el contexto de la hostelería. Estos requerimientos se han visto cubiertos al desarrollar una aplicación que permite el control dinámico de las mesas, pudiendo modificar su estado (ocupado, libre, cambio de mesa, unión de mesa, anulación de pedidos…) y el importe de sus comandas (por ejemplo, la actualización automática del precio total según se van añadiendo o eliminando productos). Del mismo modo, se ha implementado el registro de las comandas y sus movimientos, lo que podrá consultarse para el control administrativo del interesado. En definitiva, se ha desarrollado una aplicación multiplataforma adaptada al complejo mundo de la hostelería, que permite agilizar el trabajo de camareros, cocineros y administrativos, haciendo más eficiente y eficaz la dinámica de este gremio.

Respecto a las potencialidades y limitaciones de las herramientas de desarrollo escogidas, Xamarin, me ha permitido hacer aplicaciones generadas. Esto supone una gran rentabilidad del propio proceso de desarrollo, ya que, al manejar un sólo lenguaje, como es C#, el código es altísimamente reutilizable para las tres principales plataformas y la curva de aprendizaje es más rápida. Por esto, el ahorro de tiempo en el desarrollo es muy significativo. En este sentido, es destacable que Xamarin se ha descrito como la gran alternativa a los desarrollos nativos puros en la mayoría de escenarios habituales: optimiza costes y tiempos de entrega en el desarrollo multiplataforma obteniendo un resultado 100% nativo [16].

En cuanto al diseño, la arquitectura en N capas facilita el testeo de cada capa independientemente, lo que conlleva una mejor detección de errores y mantenimiento de todo el proyecto, o incluso la reutilización de capas (como es el caso de la aplicación y la web que comparten la API) [17]. Además de la escalabilidad y mejor mantenibilidad de la arquitectura, ya que dado el caso se podría escalar la base de datos con independencia de la API o al revés. Además, la ventaja más notable para el usuario es que la aplicación es mucho más ligera, ya que no es la misma aplicación móvil la que hace toda la lógica o consulta la base de datos.

En lo referente al despliegue y securización de la arquitectura, el empleo de Azure ha resultado altamente beneficioso. Al tratarse de una nube pública, cualquiera con conexión a Internet puede hacer uso de esta y al mismo tiempo dispone de una modalidad de pago por uso, que te permite compilar, implementar y administrar rápidamente aplicaciones en una red global de centros de datos de Microsoft. Mediante Azure podemos utilizar más de 240 servicios a través de internet, para aumentar nuestra productividad, disminuir costos en infraestructura, mantenimiento, personal, entre otros.

Una de las ventajas es el propio portal Microsoft Azure, en el que existen diferentes servicios de infraestructura y de plataforma para que puedas “montar” los servicios que necesites de manera sencilla, con unos cuántos clics. Además, estos servicios están garantizados con una disponibilidad del 99.99%, y en caso de fallo en disponibilidad superior, Microsoft se compromete a indemnizar por los daños [18]. Así mismo, cuenta con todas las certificaciones en materia seguridad y protección de datos, ya que, es la primera plataforma Cloud que cumple en categoría alta las características de la certificación de conformidad del Esquema Nacional de Seguridad [19].

Azure hace innecesaria la inversión en máquinas físicas y, por tanto, evita el mantenimiento que conllevan. De esta manera, se simplifica el despliegue, se agiliza el proceso de configuración y nos permite adaptar nuestros sistemas a nuestras necesidades en pocos minutos.

7.2 Trabajo futuro

Considero que además de las funcionalidades y la arquitectura desarrollada hasta el momento, el proyecto es potencialmente extensible.

Por un lado, a la aplicación se le podría añadir un módulo de gestión de proveedores para llevar la cuenta de los productos pedidos, fechas, coste… siguiendo con la línea de mejorar la eficiencia. Incluso modificar los estados de las mesas para poder gestionar reservas a determinadas horas.

En relación con la web de administración, se podría ampliar su funcionalidad, adaptándola para el cálculo y creación de estadísticas. Esto permitiría al administrador mantenerse informado sobre aspectos de gran interés como, por ejemplo, los productos más y menos vendidos o las temporadas con más ventas.

Con Application Insights se pueden generar alertas de bloqueos, uso de cada característica o tiempos, pudiendo mejorar continuamente la aplicación, hacer un seguimiento de las rutas de acceso de uso y evaluar el funcionamiento, rendimiento y éxito de cada característica [14].

Respecto a la API y base de datos, Azure permite escalabilidad de manera automatizada, por lo que un modo muy sencillo se puede añadir una directiva de crecimiento vertical de la base de datos al detectar el 80% de ocupación. Complementariamente, se podrían utilizar técnicas para agrupar las comandas (por días cada 3 meses y cada medio año, las de días por semanas…) para ahorrar espacio de memoria al aumentar los registros de la base de datos con el tiempo.

Capítulo 8. Bibliografía

[1] Instituto Nacional de Estadística, "Estructura y dinamismo del tejido empresarial en España"

https://www.ine.es/prensa/dirce_2018.pdf [Online].

[2] Mercasa, "Restauración en España. Evolución del consumo fuera del hogar"

https://www.mercasa.es/media/publicaciones/251/Mercasa_distribucion_y_consumo_154_100px.pdf [Online].

[3] WIPProject, "Desarrollo de aplicaciones móviles con Xamarin"

https://www.wip-project.com/aplicaciones-moviles-con-xamarin/ [Online].

[4] MicrosoftInsider, "Así es Visual Studio 2017, el IDE perfecto para crear apps en Windows, iOS, Android y la nube"

https://www.microsoftinsider.es/121003/asi-visual-studio-2017-ide-perfecto-crear-apps-windows-ios-android-la-nube/ [Online].

[5] Microsoft, "Introduction to the C# Language and the .NET Framework"

https://docs.microsoft.com/es-es/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework [Online].

[6] Git, "Git en Visual Studio"

https://git-scm.com/book/es/v2/Git-en-otros-entornos-Git-en-Visual-Studio [Online].

[7] Blogspot, "Arquitectura DDD y sus capas"

http://nfanjul.blogspot.com/2014/09/arquitectura-ddd-y-sus-capas.html [Online].

[8] Microsoft, "Creating mobile apps with Xamarin.Forms"

https://docs.microsoft.com/es-es/xamarin/xamarin-forms/creating-mobile-apps-xamarin-forms/ [Online].

[9] Blog de José M. Aguilar, "ASP.NET Core 2.1"

https://www.variablenotfound.com/2018/06/novedades-de-aspnet-core-21.html [Online].

[10] clavei, "¿Qué aporta MVC al desarrollo de aplicaciones Web?"

https://www.clavei.es/blog/que-aporta-mvc-al-desarrollo-de-aplicaciones-web/ [Online].

[11] Deloitte, "¿Qué es un ORM?"

https://www2.deloitte.com/es/es/pages/technology/articles/que-es-orm.html [Online].

[12] Microsoft Azure, "SQL Database"

https://azure.microsoft.com/es-es/services/sql-database/ [Online].

[13] Microsoft Azure, "¿Qué es PaaS?"

https://azure.microsoft.com/es-es/overview/what-is-paas/ [Online].

[14] Encamina, "Monitorizando tu Aplicación en Azure con Application Insights"

https://blogs.encamina.com/por-una-nube-sostenible/monitorizando-tu-aplicacion-en-azure-con-application-insights/ [Online].

[15] Microsoft, "What is Application Insights?"

https://docs.microsoft.com/es-es/azure/azure-monitor/app/app-insights-overview [Online].

[16] Xataka, "Xamarin: la gran candidata a ganadora del desarrollo de aplicaciones para móvil"

https://www.xataka.com/aplicaciones/xamarin-la-gran-candidata-a-ganadora-del-desarrollo-de-aplicaciones-para-movil [Online].

[17] iComparable, "¿Arquitectura N-Tier o Arquitectura N-Layer?"

http://icomparable.blogspot.com/2008/10/arquitectura-n-tier-o-arquitectura-n.html [Online].

[18] Microsoft Azure, "Resumen de SLA para los servicios de Azure"

https://azure.microsoft.com/es-es/support/legal/sla/summary/ [Online].

[19] Tecon, "¿Qué es Microsoft Azure? ¿Cómo funciona?"

https://www.tecon.es/que-es-microsoft-azure-como-funciona/ [Online].