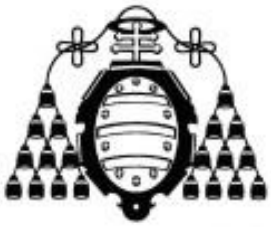
UNIVERSIDAD DE OVIEDO



ESCUELA DE INGENIERÍA INFORMÁTICA

PROYECTO FIN DE MÁSTER

"Savour: Aplicación Web adaptada a dispositivos móviles para la gestión de pedidos hosteleros con componentes CRM y de pago virtual"

DIRECTOR: Alberto Manuel Fernández Álvarez

AUTOR: Pablo Soto Medina

Vº Bº del Director del Proyecto

Agradecimientos

No me gustaría dejar pasar la oportunidad de mostrar mi agradecimiento a todas aquellas personas que colaboraron de una forma u otra a la concreción de este proyecto:

- A Elena, prometida y futura esposa, por soportar muchos fines de semana de trabajo, por su apoyo incondicional y sus ánimos cuando más lo necesitaba.

- A mis padres, Francisco y Brígida y a mi hermano Javier. Por su cariño e interés en mí y este proyecto. Por todas las cosas que me han dado y por las que siempre estaré agradecido.

- A Jose Manuel García, tío y padrino, por su consejo y ayuda desinteresada en este proyecto, fruto de su amplia experiencia en el sector hostelero.

- A Alberto Manuel Fernández Álvarez, por creer de nuevo en una de mis ideas abstractas, por ayudarme a darle forma, y dedicar parte de su tiempo a guiarme y orientarme.

- A los compañeros que he conocido durante estos años en la universidad, de los que muchos se han convertido en amigos de verdad.

- A todos aquellos que han contribuido, directa o indirectamente, en el desarrollo de este proyecto y, por despiste u omisión, no aparecen reflejados en estas líneas.

A todos ellos, Gracias.

Resumen

La tecnología avanza muy rápido en casi todos los ámbitos. Hoy día no resulta extraño reservar un parking a través de un dispositivo móvil, o incluso abrir la barrera escaneando un código QR. Además, utilizar aplicaciones por parte de los clientes en detrimento de los procesos tradicionales, adquiere un beneficioso componente social y de análisis para el negocio, mediante sistemas de comentarios y feedback, estadísticas, publicidad en redes sociales, etc.

El presente proyecto pretende utilizar tecnologías web adaptables a dispositivos móviles para sustituir las TPV tradicionales y adecuar los procesos de negocio hosteleros a el uso de aplicaciones web reactivas, con componentes CRM y de estadísticas, con el fin de automatizar procesos, obtener un mayor control sobre el negocio, involucrar y fidelizar a los clientes, y beneficiarse de lo que ello conlleva.

Palabras Clave

Hosteleria, TPV, Carta virtual, CRM, Aplicación web.

Abstract

Technology moves quickly in almost every field. Nowadays, it does not seem surprising to reserve parking through a mobile phone, or even open the gate scanning a QR code. Besides, using apps by customers instead of traditional methods, acquires an important social component as well as for business analysis, by the use of rating and feedback systems, stats, adds on social networks, etc.

The purpose of this proyect is to use mobile-adaptative web technologies in order to replace tradicional POS and adapt restoration business for the use of reactive web applications with CRM and Stats components, in order to automate processes, gain greater control over business, engage and retain customers, and benefit from what it entails.

Keywords

Restoration, POS, Virtual menu, CRM, Web app.

Índice General

Índice de Figuras

Capítulo 1. Introducción

1.1 Justificación del Proyecto

La hostelería y turismo son uno de los pilares fundamentales para el sustento económico de este país. No cabe duda de que el sector ha sufrido una evolución positiva en los últimos años, pero es cierto que lo ha hecho más bien desde el punto de vista culinario y de infraestructuras que desde el tecnológico.

La gran mayoría de estos negocios utilizan TPVs táctiles sobre sistemas Windows ejecutando una o varias aplicaciones de escritorio generalmente no distribuidas, lo cual dificulta las tareas de facturación y contabilidad al tener que unificar datos provenientes de varios equipos.

Algunas de las soluciones distribuidas más avanzadas ofrecen terminales móviles al personal de mesa, haciendo que las órdenes se generen en el mismo momento en el que los clientes las solicitan, tratando de agilizar los procesos. En la práctica no siempre surte el efecto deseado pues siguen existiendo muchas tareas manuales como hacer llegar las órdenes a cocina o la priorización de pedidos.

Por otro lado, a diferencia de otros sectores, la hostelería cuenta con muy pocos mecanismos de evaluación, promoción y fidelización de clientes, haciendo muy complicado para los negocios evaluar la satisfacción de sus clientes, que suelen utilizar plataformas externas para expresar su opinión en lugar de hacerla llegar directamente a los responsables del mismo.

Tampoco suelen ofrecerse servicios de pago virtual o de pago anticipado, muy utilizados por ejemplo en reservas de transporte, museos, cine, etc. Que podrían atraer nuevos clientes o mejorar la satisfacción de los actuales.

Todos estos motivos se consideran suficientes para el desarrollo del presente proyecto.

1.2 Objetivos del Proyecto

El propósito final de este proyecto, es construir una aplicación de gestión para pedidos hosteleros que cumpla los siguientes objetivos:

- Naturaleza distribuida basada en tecnologías web.
- Adaptación a dispositivos móviles.
- Identificación de la mesa y carta virtual.
- Proporcionar mecanismos de promoción y fidelización.
- Servicios de pago virtual.
- Consulta de estadísiticas en tiempo real.
- Fácilmente escalable.

1.3 Estudio de la Situación Actual

El estudio se ha dividido en varios frentes para una mejor evaluación y clasificación de las alternativas existentes. En primer lugar, se han estudiado las soluciones integrales que se ofrecen en España, posteriormente se han comparado con otras soluciones integrales de otros países y por último se han evaluado únicamente aquellas soluciones basadas en tecnologías web junto con aplicaciones móviles [1].

1.3.1 Soluciones integrales para hostelería

Existen numerosas empresas en España que se dedican a realizar software de gestión para negocios hosteleros. La mayoría de soluciones que se han encontrado incluyen los siguientes elementos:

- Equipo con pantalla táctil y sistema operativo Windows.
- Software de gestión con varios módulos (generalmente programado en Basic)
- Impresora térmica de tickets
- Caja fuerte

Como elementos opcionales se suelen añadir:

- Aplicaciones móviles para PDA's o dispositivos Android que se comunican con el software de gestión.
- Sistemas de reservas web
- Otros módulos a medida

Algunos ejemplos de estas empresas pueden ser HostelTáctil, Infotronic, Emesa, ICG, TPVFácil, etc.

Ilustración 1. Conjunto típico de elementos ofertados en soluciones integrales

En el extranjero la situación no cambia demasiado, la mayoría de soluciones siguen siendo aplicaciones de escritorio poco estéticas visualmente y en gran medida obsoletas, sin embargo, si se aprecian varias empresas que comienzan a utilizar tecnologías web y dispositivos móviles Android e iOS como complemento.

Algunas de las empresas estudiadas son ToastPOS, TouchBistro, Gotmerchant y BePoz.

Ilustración 2: GoMerchant

1.3.2 Soluciones basadas en tecnologías web

Dado que el presente proyecto utiliza como base las tecnologías web, se ha realizado especial hincapié en la búsqueda de soluciones que utilicen esta tecnología.

A continuación, se describen varias de las soluciones encontradas que han resultado más completas en su conjunto.

Lightspeed Restaurant

Lightspeed es una de las soluciones que más éxito están teniendo actualmente en negocios sobretodo de los Estados Unidos. Utiliza servicios web para la comunicación entre las aplicaciones y el servidor, quedando todos los datos sincronizados tanto en el dispositivo como en la nube.

Cuenta con numerosas características, desde la personalización de la carta, hasta el servicio de pagos, pasando por otras tan importantes como la gestión de reservas, estadísticas, fidelización, pedidos remotos, etc.

Es compatible únicamente con dispositivos Apple (Mac, iPad y iPhone), siendo probablemente la gran desventaja de la solución, pues es una de las más completas en cuanto a características. Utiliza aplicaciones nativas en cada una de las plataformas.

Ilustración 3: Lightspeed restaurant

Cashera

Se trata de una de las pocas soluciones de punto de venta totalmente web. Cuenta con un diseño moderno y adaptable a múltiples pantallas y dispositivos. Además, cuenta con aplicaciones móviles híbridas que muestran parte del contenido utilizando webviews.

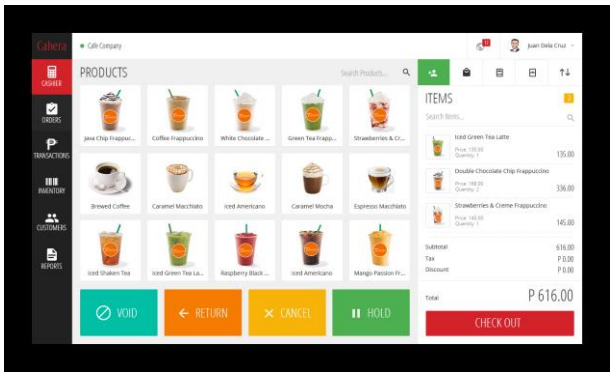


Ilustración 4- Cashera

En cuanto a su funcionalidad, decir que se trata de una solución genérica y al no estar centrada exclusivamente en la hostelería se echan de menos muchas características interesantes como la gestión de mesas, gestión de reservas, pedidos remotos, etc.

Aún así mantiene el resto de características básicas como gestión de productos, gestión de pagos, inventarios, estadísticas, etc. Al igual que Lightspeed cuenta con sincronización en la nube.

Kounta

Kounta es una solución australiana totalmente web y adaptable a dispositivos móviles enfocada exclusivamente en la hostelería y restauración.

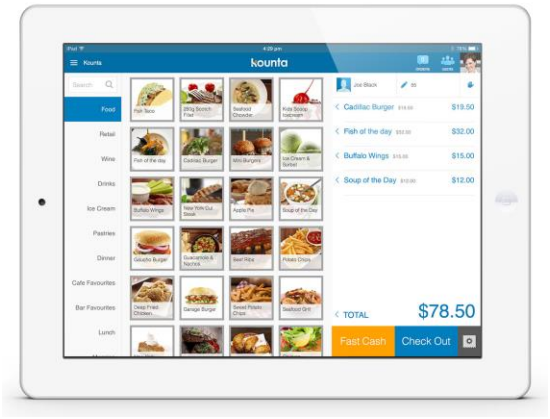


Ilustración 5: Kounta

Es probablemente la solución web más completa que se ha encontrado, contando con características como gestión de mesas, fidelización, sincronización en la nube, gestión de pagos, estadísticas, etc.

Simplemente es necesario un navegador web, o alguna de las aplicaciones híbridas que dispone para utilizarlo. Quizás lo único que se echa en falta sea una interfaz para los clientes de modo que puedan realizar reservas o pedidos desde sus propios dispositivos, ya que por el momento la solución sólo es accesible por el propio staff del negocio.

1.3.3 Otras soluciones

Durante el proceso de estudio se han encontrado otras soluciones que nada tienen que ver con las anteriores pero que merece la pena resaltar. Un ejemplo claro es **Foozaps Restaurant POS**, una aplicación nativa Android que se utiliza principalmente en tablets como TPV.

La gran ventaja de Foozaps es que se trata de una aplicación gratuita, con algunos servicios Premium, por lo que resulta muy interesante para pequeños negocios que acaben de empezar su actividad.

Otra solución interesante puede ser **Revel iPad POS**, que como su nombre indica, está orientada únicamente a dispositivos Apple, en especial el iPad. De forma similar a muchas de las anteriores también cuenta con sincronización en la nube, y tiene un gran número de características que sin duda la hacen una solución muy completa.

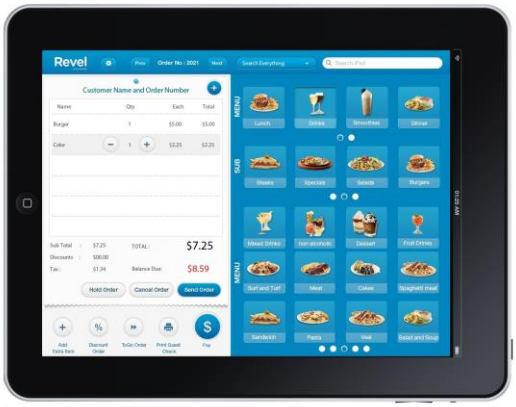


Ilustración 6: iPad revel POS

Como siempre, la gran pega de este tipo de aplicaciones es no disponer de un módulo de comunicación para los clientes, de forma que puedan hacer pedidos desde sus propios dispositivos, pagar, reservar o realizar comentarios, además de no ser multiplataforma y por tanto limitar su uso únicamente a dispositivos Apple.

1.3.4 Evaluación de Alternativas y Justificación de la elección

Llegados a este punto, tras estudiar las soluciones existentes y teniendo claro que el proyecto será una aplicación web adaptable a dispositivos móviles mediante plataformas híbridas, queda definir qué características harán diferentes a Savour con respecto al resto de soluciones del mercado, y por tanto justiciar el desarrollo del presente proyecto.

A continuación, se muestra un cuadro con una comparativa entre las características deseadas en el presente proyecto, y las correspondencias con las soluciones actuales.

Característica	Savour	Lightspeed	Cashera	Kounta	Foozaps	Revel
Multiplataforma	**X**		X	X		
Sincronización en la nube	**X**	X	X	X	X	X
Carta virtual	**X**	X		X	X	X
Generación facturas	**X**	X	X	X	X	X
Apps móviles para clientes	**X**					
CRM	**X**	X	X	X		X
Estadísticas	**X**	X	X	X	X	X

Tabla: Comparativa de características entre diferentes soluciones

Como se puede observar, ninguna de las soluciones estudiadas cumple completamente con las características propuestas para este proyecto. En especial el disponer de una interfaz de la aplicación orientada a los clientes del local, característica que se estima indispensable para que los propios clientes puedan realizar pedidos, pagar o realizar comentarios sobre su experiencia desde sus propios dispositivos.

1.4 Presentación de la idea

El proyecto cuenta con 2 subsitemas bien diferenciados. Una **aplicación web de gestión del restaurante** para el negocio, y otra **aplicación móvil para los clientes**, comunicándose ambas con el mismo backend.

A continuación, se describe de forma sencilla y a escala global el funcionamiento de ambas aplicaciones y de sus principales funciones.

Ilustración 7 - Logo de la aplicación

1.4.1 Aplicación para el negocio

La aplicación para el negocio, está desarrollada con Ionic y por tanto puede utilizarse directamente sobre cualquier navegador web, o también a través de las aplicaciones móviles que permite generar el framework.

Esta aplicación viene a sustituir las **TPV del negocio**, contanto con **4 vistas** bien diferenciadas:

Vista de sala

Muestra el **estado actual de la sala** y un listado de todas las comandas activas, con un resumen para cada una de ellas de una forma sencilla y visual. A su vez, permite añadir nuevas comandas, editar las existentes y realizar cobros.

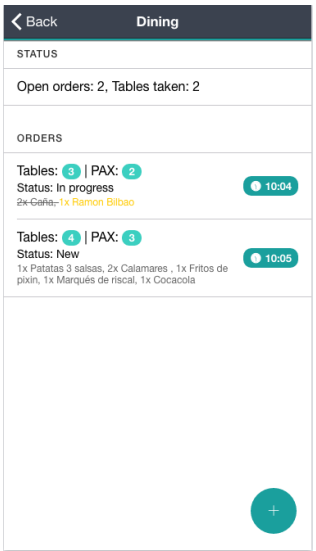


Ilustración 8 - Vista de sala

El maître o los camareros serían los encargados de operar con esta vista.

Vista de barra

Todas las **bebidas** de las comandas aparecen en esta vista por orden de prioridad en el momento de su creación. De esta manera, los responsables de barra podrán ir preparando la bebida en el mismo instante en el que la solicite el cliente, pudiendo avisar al camarero en el momento en el que esté preparada.

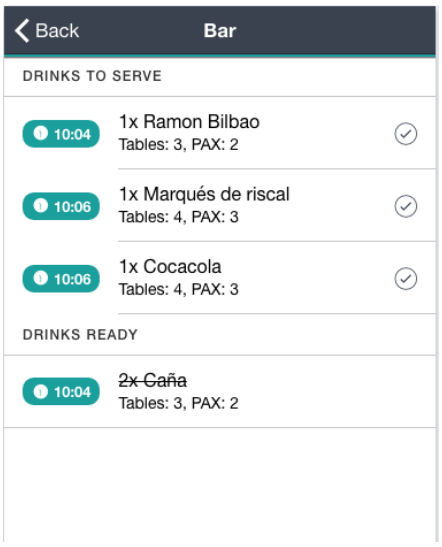


Ilustración 9 - Vista de barra

Vista de cocina

De forma análoga a la vista de barra, toda la comida ordenada pasará por esta vista en el momento en el que los clientes la soliciten. En este caso, los cocineros serían los responsables de la elaboración y aviso a la sala en el momento en el que esté preparada.

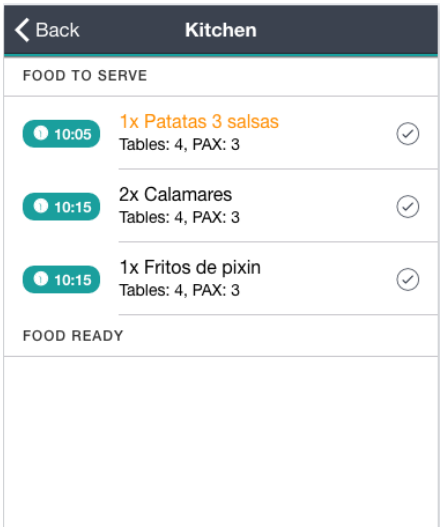


Ilustración 10 - Vista de cocina

Vista de administración

Más que una vista, podría considerarse como un **menú de administración**. A través del mismo podría accederse a:

- **Gestión de productos**: Para la creación, edición o eliminación de los productos de la carta.
- **Gestión de usuarios**: Para la creación y modificación de los usuarios que pueden acceder al sistema.
- **Estadísticas**: contiene varias tablas de estadísticas con información relevante sobre el estado del negocio.
- **CRM**: Desde esta pantalla puede seguirse la evolución de la fidelización con los clientes además de leer críticas y comentarios.

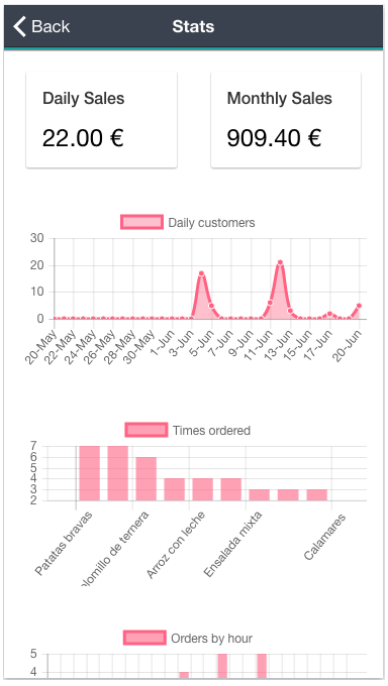


Ilustración 11 - Pantalla de estadísticas

1.4.2 Aplicación para el cliente

Una de las grandes ventajas del proyecto consiste en permitir a los clientes realizar pedidos de forma opcional, sin necesidad de esperar y establecer contacto con los camareros.

De esta manera, cuando un cliente llega al local y se sienta en una mesa, se encontrará una pegatina similar a la que se muestra a continuación:

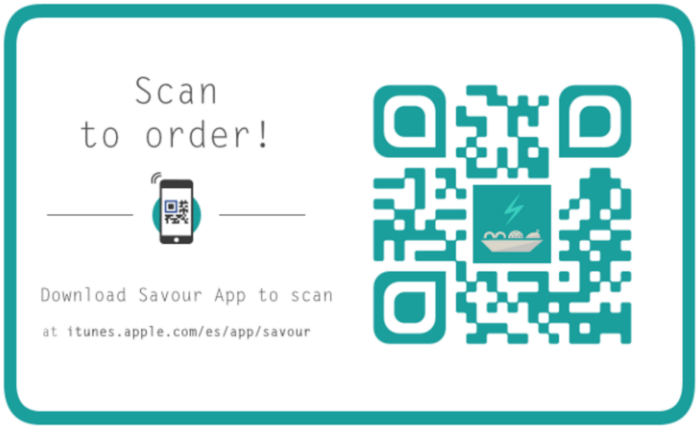


Ilustración 12 - Ejemplo de pegatina QR

Con ella, el cliente sabrá que, si lo desea, podrá realizar pedidos descargándose la aplicación y escaneando el código QR que se muestra.

Dicho código **identifica** internamente el **restaurante** y la **mesa** donde está sentado el comensal, por lo que contiene toda la información para mostrar la carta correspondiente al local y realizar una orden.

Los clientes que realicen los pedidos de esta forma, podrán además **conocer en tiempo real el estado actual** de preparación, llamar al camarero o solicitar la cuenta.

Finalmente, una vez realizado el cobro, podrán **valorar su experiencia** y volver de nuevo a realizar otra orden.

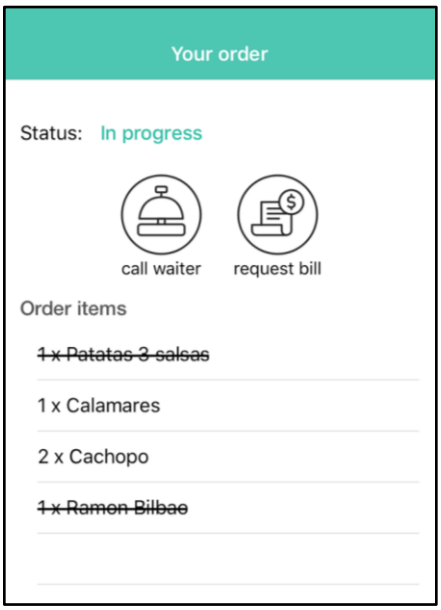


Ilustración 13 - Estado del pedido

Capítulo 2. Aspectos Teóricos

En este capítulo se describirán aquellos aspectos teóricos referentes a los servicios de hostelería y restauración que se consideran relevantes para el desarrollo del proyecto.

Hay que tener en cuenta que, para lograr una solución útil para este sector, esta debe adaptarse a sus procesos de forma natural, tratando de minimizar los cambios en las tareas habituales del negocio y por tanto el aprendizaje del personal.

2.1 Mise en place

Del francés "puesto en su lugar", se emplea en gastronomía para designar la tarea de colocación y ordenación de ingredientes, material, especias, etc, que el cocinero vaya a emplear en el proceso de elaboración de los alimentos.

Traducido al personal de comedor, se trata del proceso de disposición de mesas, cubertería, mantelería, etc.

Muchos restaurantes, sobretodo en lugares turísticos o de menú diario, realizan este proceso antes de la llegada de los clientes, y en función de lo que ordenen realizan los cambios de cubertería pertinentes. El resto sin embargo entregan la carta en primer lugar y realizan este proceso a posteriori.

Para la presente solución se recomienda **realizar el mise en place de la mesa después del servicio de carta**, pues la cubertería podría dificultar a los clientes utilizar un dispositivo móvil o tablet para realizar sus pedidos [2] [4].

2.2 Organización de los procesos de sala

El servicio de sala comprende una serie de operaciones que pueden dividirse en las siguientes fases [3]:

1. Antes del servicio
2. Durante el servicio
3. Después del servicio

Antes del servicio

Comprende todas las actividades que deben realizarse antes de la entrada de los clientes al comedor. Algunas de estas actividades son:

- Mise en place del local y cocina
- Repaso del libro de reservas
- Distribución de rangos a los componentes de la brigada
- Preparación de agua, pan y vino
- Preparación de aperitivos
- Explicación de la composición del menú y de los posibles cambios a los componentes de la brigada

Durante el servicio

Esta fase incluye todas las actividades desde la llegada del cliente hasta su salida.

La recepción del cliente puede ser realizada por distintos componentes de la brigada, aunque generalmente esta tarea corresponde al jefe de sala. En locales suficientemente grandes puede existir la figura de recepcionista y el servicio de ropero.

Una vez que el cliente ha sido recibido y acomodado, el siguiente paso sería la presentación de la carta, siempre abierta y por la derecha.

Es en este momento cuando se debería presentar al cliente la **posibilidad de utilizar un dispositivo móvil propio o proporcionado por el local, como carta virtual** o proceder con el servicio tradicional, donde sería el camarero el encargado de utilizar dicho dispositivo para anotar la comanda.

Una vez realizada la comanda, los siguientes pasos del servicio serían:

- Mise en place de la mesa
- Servir aperitivo
- Servir pan
- Servir vino
- Retirar aperitivo
- Servir entrantes
- Repasar agua, vino y pan
- Servir platos principales

- Retirar platos principales y cubiertos
- Retirar plato de pan
- Repasar agua
- Retirar migas de la mesa
- Tomar comanda de los postres
- Servir postres
- Retirar postres
- Retirar vino
- Tomar comanda de café y licores
- Servir café y licores
- Retirar cafés
- Pasar factura

Cuando el cliente abandone el local, únicamente debe quedar en la mesa la copa de agua, la servilleta y el vaso del licor si lo hubiese.

Después del servicio

La tercera fase del servicio de sala comprende las tareas realizadas después del servicio. En la mayoría de ocasiones estas tareas se realizan una vez que el cliente ha habandonado la sala, pero en algunos establecimientos pueden realizarse aún con los clientes dentro.

En esta fase es muy importante seguir con extrema pulcritud todos los pasos. Algunas de las normas básicas son:

- Utilizar una bandeja para retirar el material.
- Evitar transportar las copas con la mano, sin introducir los dedos en las mismas.
- Retirar las servilletas una a una tomándolas por un pico.
- Retirar el mantel sin que caigan migas al suelo.
- Nunca retirar el mantel mientras lo clientes permanezcan en el comedor.

2.3 La comanda

La comanda es probablemente el concepto más importante para el desarrollo de este proyecto pues definirá el flujo de ejecución y modelo de estados para cada pedido realizado.

La palabra "comanda" proviene de un galicismo de la palabra "commander", que podríamos traducir como "pedir". Se trata de un vale de recorrido interno que permite conocer, elaborar y servir el pedido del cliente en tiempo y forma. Además, justifica los movimientos de materias primas utilizados, lo que permite la generación de la factura y el control del flujo de géneros [5][6].

2.3.1 Formato

Se trata de un papel de color blanco autocopiativo, generalmente con una copia de color rosado y otra amarilla. El formato suele variar según la naturaleza del servicio, pero al menos debe contener los siguientes elementos:

1. Numeración de la comanda
2. Fecha y número de mesa
3. Número de habitación (en el caso de los hoteles)
4. Número de comensales
5. Nombre del camarero
6. Listado de conceptos (platos o elementos solicitados).
7. Tiempos de servicio/pase: Consiste en señalar mediante una línea horizontal cuando deben pasarse los platos demandados.

Ilustración 14 - Tipos de comanda

2.3.2 Itinerario y fases

El movimiento de una comanda puede representarse por los siguientes pasos:

1. El encargado toma la comanda por triplicado, guardando una copia sellada para facturación y otra copia para cocina, o la zona de elaboración. La tercera copia se guarda para el servicio. Este procedimiento puede variar ligeramente en función del negocio.

2. Al entregar la copia de la comanda en cocina, se cantará la siguiente voz: "Marcha comanda mesa X", que indicará a los cocineros y ayudantes de cocina que deben iniciar la elaboración de un nuevo pedido.

3. Una vez que esté disponible el primer plato solicitado, se cantará: "Empieza la mesa X".

4. Para los platos posteriores se cantará: "Sigue la mesa X".

5. Al finalizar, con el último plato, se cantará: "Termina la mesa X".

Durante todo este proceso, después de servir cada plato, se deberá realizar el marcado de los platos servidos en la comanda, con el fin de conocer el estado de la mesa en todo momento.

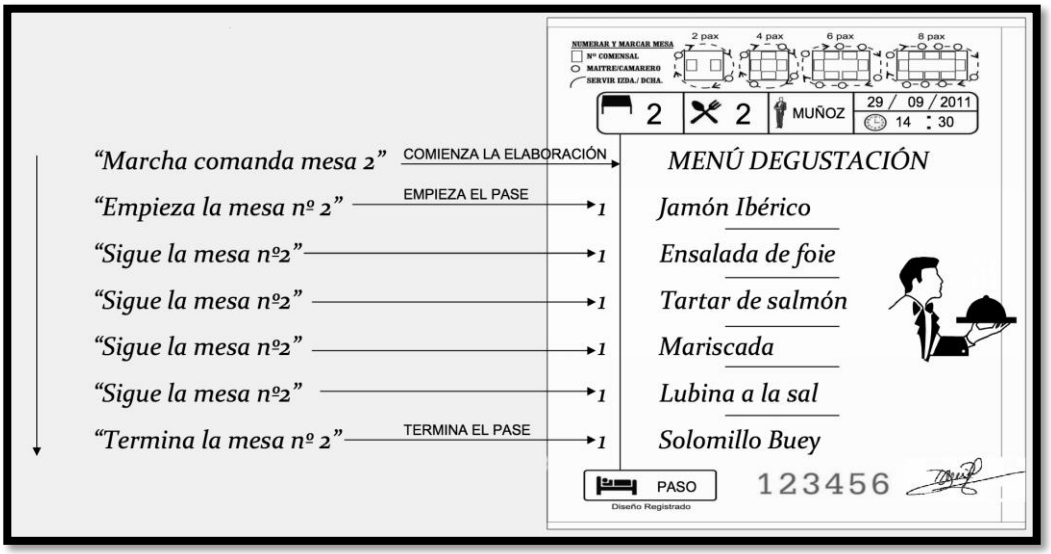


Ilustración 15 - Ejemplo de comanda y fases

2.3.3 Comandas especiales

Son comandas no convencionales, generalmente de comedor, tomadas en situaciones contretas. A continuación, se enumeran varias de estas comandas:

- **Retour o devolución**: Se utiliza para indicar que lo que figura en la comanda ha sido devuelto por el cliente. En el caso de no haber sido preparado aún, será necesario comunicarlo rápidamente a cocina, o supondrá el reingreso de un plato. Para facturación o caja, servirá para indicar que el plato, bebida o postre deberá ser descontado de la factura.

- **Suite o seguido**: Se puede utilizar en 2 casos. Cuando hay nuevos comensales que se incorporen a una mesa en la cual ya se hubiese solicitado una comanda, o cuando los clientes soliciten un nuevo plato, bebida o postre, que por lo tanto deba ser incorporado a la factura.

- **En place o en lugar de**: Se trata de una devolución que implica un cambio de un plato por otro, cuando el cliente cambia de opinión una vez que ya hizo su pedido.

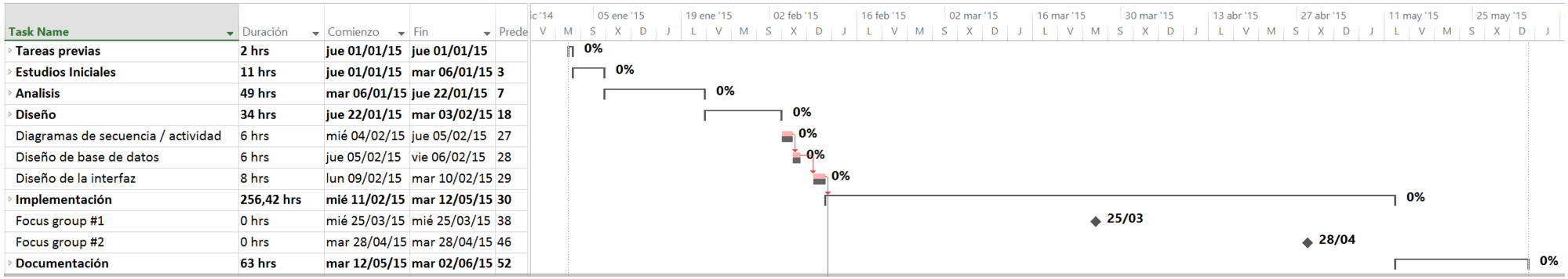
- **V.I.P**: Se utiliza cuando quiera darse un trato privilegiado a una mesa por cuestiones diversas. Este servicio especial debe ser sumamente discreto para evitar agravios comparativos.

- **Anotaciones de elaboración**: Sobretodo en el caso de carnes, se pueden añadir ciertas siglas para indicar el punto de cocción solicitado por el cliente. Algunos ejemplos son: M.P.H (muy poco hecho), P.H (poco hecho), H (hecho), M.P (muy pasado).

Capítulo 3. Planificación del Proyecto y Presupuesto

3.1 Planificación inicial

A continuación, se muestra el diagrama de Gantt asociado a la primera planificación que se realizó del proyecto.



Task Name	Duración	Comienzo	Fin	Prede
Tareas previas	**2 hrs**	**jue 01/01/15**	**jue 01/01/15**	
Estudios Iniciales	**11 hrs**	**jue 01/01/15**	**mar 06/01/15**	**3**
Analisis	**49 hrs**	**mar 06/01/15**	**jue 22/01/15**	**7**
Diseño	**34 hrs**	**jue 22/01/15**	**mar 03/02/15**	**18**
Diagramas de secuencia / actividad	6 hrs	mié 04/02/15	jue 05/02/15	27
Diseño de base de datos	6 hrs	jue 05/02/15	vie 06/02/15	28
Diseño de la interfaz	8 hrs	lun 09/02/15	mar 10/02/15	29
Implementación	**256,42 hrs**	**mié 11/02/15**	**mar 12/05/15**	**30**
Focus group #1	0 hrs	mié 25/03/15	mié 25/03/15	38
Focus group #2	0 hrs	mar 28/04/15	mar 28/04/15	46
Documentación	**63 hrs**	**mar 12/05/15**	**mar 02/06/15**	**52**

Ilustración 16 - Diagrama de gantt con la primera planificacion

Descripción

Se trata de la primera planificación elaborada durante la asignatura de "Dirección y gestión de proyectos". En este momento se realizó una planificación ideal, como si el desarrollo del proyecto comenzase al finalizar la docencia (enero de 2015), a pesar de que ya se sabía que el proyecto se pospondría al menos hasta el año siguiente, por motivos profesionales.

La duración estimada era de 435 horas, decidiéndose repartir en jornadas de 20 horas semanales, lo que suponía un tiempo de desarrollo de unos 6 meses.

3.2 Segunda planificación

Este es el diagrama de Gantt para la segunda planificación:

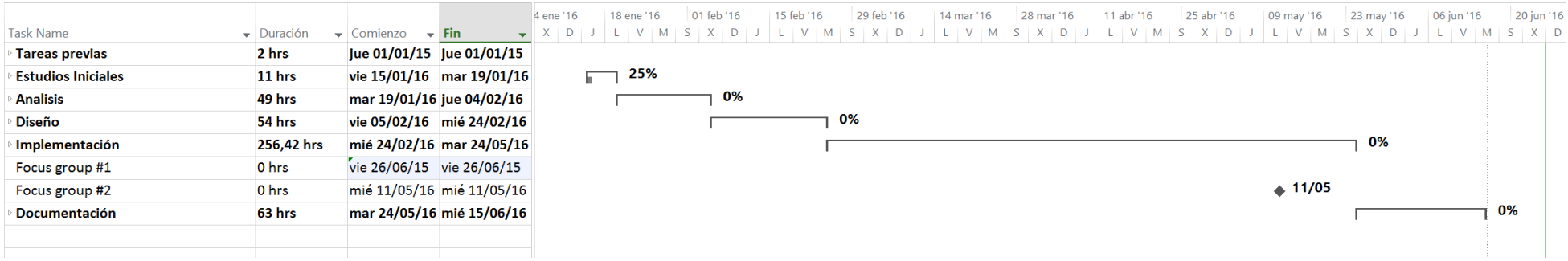


Ilustración 17 - Diagrama de gantt con la segunda planificación

Descripción

A sabiendas de que en 2015 sería imposible dedicar suficiente tiempo al proyecto, se decide realizar una nueva planificación comenzando el 15 de enero de 2016. En este punto, ya se habían realizado varias tareas como los **estudios iniciales**, y el **primer focus group** con una persona del sector.

3.3 Desarrollo real del proyecto

A continuación, se muestra el diagrama de Gantt correspondiente al desarrollo final y real del proyecto.

Nombre de tarea	Comienzo	Fin	Tiempo empleado
Tareas previas	jue 01/01/15	vie 30/01/15	9h
Estudios iniciales 1	sáb 28/02/15	vie 20/03/15	10h
Estudios iniciales 2	vie 15/01/16	sáb 16/01/16	6h
Análisis	sáb 27/02/16	mié 22/06/16	44h
Diseño	mar 07/06/16	lun 20/06/16	19h
Implementación	dom 01/05/16	sáb 25/06/16	104h
Documentación 1	lun 01/02/16	dom 07/02/16	13h
Documentación 2	mar 24/05/16	mar 28/06/16	36h
Focus group #1	vie 26/06/15	vie 26/06/15	2h
Focus group #2	jue 23/06/16	jue 23/06/16	2h

Ilustración 18 - Diagrama de Gantt con el desarrollo real del proyecto

Nombre de tarea	Comienzo	Fin	Tiempo empleado
Tareas previas	jue 01/01/15	vie 30/01/15	9h
Estudios iniciales 1	sáb 28/02/15	vie 20/03/15	10h
Estudios iniciales 2	vie 15/01/16	sáb 16/01/16	6h
Análisis	sáb 27/02/16	mié 22/06/16	44h
Diseño	mar 07/06/16	lun 20/06/16	19h
Implementación	dom 01/05/16	sáb 25/06/16	104h
Documentación 1	lun 01/02/16	dom 07/02/16	13h
Documentación 2	mar 24/05/16	mar 28/06/16	36h
Focus group #1	vie 26/06/15	vie 26/06/15	2h
Focus group #2	jue 23/06/16	jue 23/06/16	2h

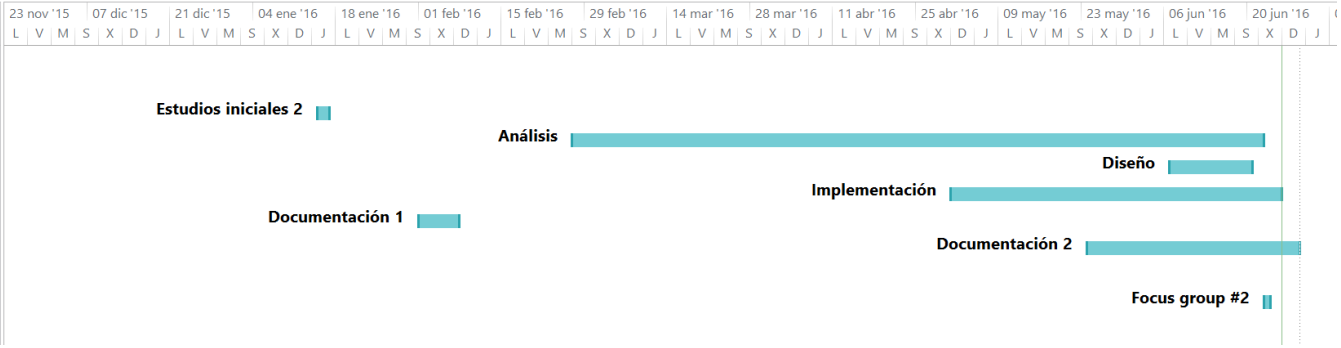


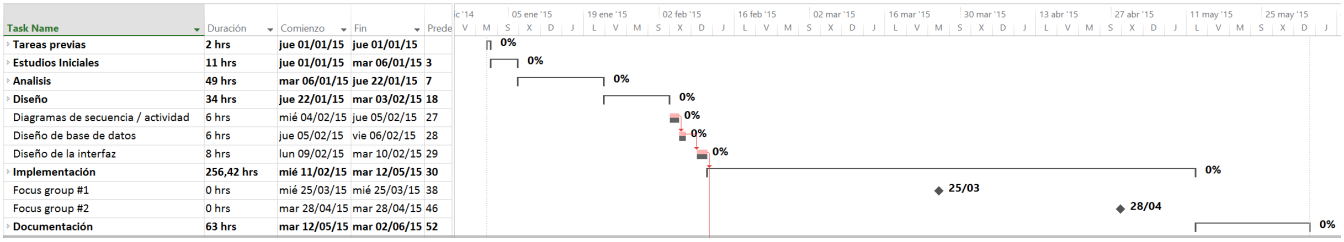
Ilustración 19 – Diagrama de Gantt con el desarrollo final del proyecto (zoom sobre 2016)

Se muestra una vista general del desarrollo global junto con la vista ampliada a 2016.

3.4 Comparativa entre planificaciones y desarrollo real del proyecto

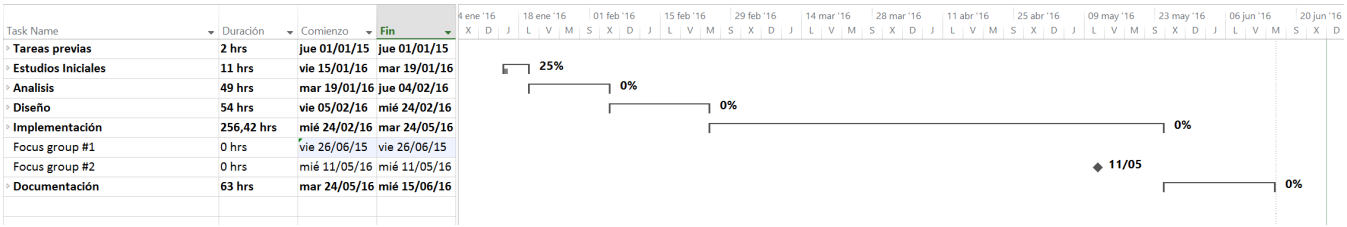
A continuación, se muestra una comparativa entre las planificaciones utilizadas en el desarrollo de este proyecto.

Planificación inicial



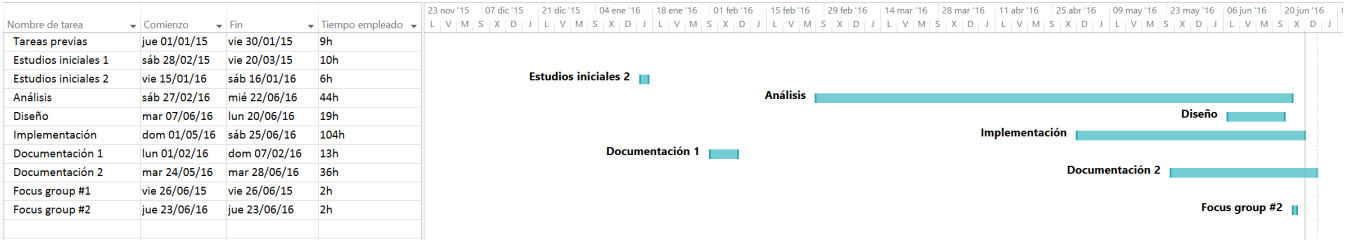
Task Name	Duración	Comienzo	Fin	Prede
Tareas previas	**2 hrs**	**jue 01/01/15**	**jue 01/01/15**	
Estudios Iniciales	**11 hrs**	**jue 01/01/15**	**mar 06/01/15**	**3**
Analisis	**49 hrs**	**mar 06/01/15**	**jue 22/01/15**	**7**
Diseño	**34 hrs**	**jue 22/01/15**	**mar 03/02/15**	**18**
Diagramas de secuencia / actividad	6 hrs	mié 04/02/15	jue 05/02/15	27
Diseño de base de datos	6 hrs	jue 05/02/15	vie 06/02/15	28
Diseño de la interfaz	8 hrs	lun 09/02/15	mar 10/02/15	29
Implementación	**256,42 hrs**	**mié 11/02/15**	**mar 12/05/15**	**30**
Focus group #1	0 hrs	mié 25/03/15	mié 25/03/15	38
Focus group #2	0 hrs	mar 28/04/15	mar 28/04/15	46
Documentación	**63 hrs**	**mar 12/05/15**	**mar 02/06/15**	**52**

Segunda planificación



Task Name	Duración	Comienzo	Fin
Tareas previas	**2 hrs**	**jue 01/01/15**	**jue 01/01/15**
Estudios Iniciales	**11 hrs**	**vie 15/01/16**	**mar 19/01/16**
Analisis	**49 hrs**	**mar 19/01/16**	**jue 04/02/16**
Diseño	**54 hrs**	**vie 05/02/16**	**mié 24/02/16**
Implementación	**256,42 hrs**	**mié 24/02/16**	**mar 24/05/16**
Focus group #1	0 hrs	vie 26/06/15	vie 26/06/15
Focus group #2	0 hrs	mié 11/05/16	mié 11/05/16
Documentación	**63 hrs**	**mar 24/05/16**	**mié 15/06/16**

Desarrollo real del proyecto



Nombre de tarea	Comienzo	Fin	Tiempo empleado
Tareas previas	jue 01/01/15	vie 30/01/15	9h
Estudios iniciales 1	sáb 28/02/15	vie 20/03/15	10h
Estudios iniciales 2	vie 15/01/16	sáb 16/01/16	6h
Análisis	sáb 27/02/16	mié 22/06/16	44h
Diseño	mar 07/06/16	lun 20/06/16	19h
Implementación	dom 01/05/16	sáb 25/06/16	104h
Documentación 1	lun 01/02/16	dom 07/02/16	13h
Documentación 2	mar 24/05/16	mar 28/06/16	36h
Focus group #1	vie 26/06/15	vie 26/06/15	2h
Focus group #2	jue 23/06/16	jue 23/06/16	2h

Ilustración 20 - Comparativa entre planificaciones

Conclusión

El desarrollo real del proyecto vuelve a sufrir una grave desviación con respecto a la segunda planificación por motivos laborales y personales. Para tratar de finalizar el proyecto antes de la convocatoria deseada, a mediados de febrero de 2016 se toman varias decisiones que expongo a continuación:

- Incluir todos los documentos susceptibles de aparecer en esta documentación, directamente sobre la pantilla final. Esto finalmente ahorra un tiempo considerable al no tener que juntar y formatear varios documentos.

- Descartar la implementación de uno o varios prototipos, desarrollando directamente al producto final. Esto ha supuesto un ahorro de aproximadamente 100 horas en el desarrollo, aunque también supuso tener que dedicar un par de días a refactorizar código, para limpiarlo y adaptarlo a los nuevos elementos de análisis y diseño que se iban generando.

- Elaboración de tareas en paralelo. Evitando una planificación en cascada se consiguió centrar el foco en tareas sobre el mismo concepto, aunque en diferentes niveles de la cascada. Por ejemplo, el diseño de la base de datos se elaboraba al mismo tiempo que se implementaba.

3.5 Resumen de tareas

Además de las planificaciones realizadas a lo largo de este proyecto, se ha llevado un **seguimiento diario de las tareas**, que se puede consultar íntegramente en el apartado 10.1 del presente documento.

Una visión reducida de dicho diario se muestra en la siguiente tabla, donde se recoge el tiempo empleado en cada una de las fases del proyecto.

Categoría	Horas
Tutoría	5
Formación	21
Análisis	64
Diseño	19
Implementación	120
Pruebas	12
Documentación	52
Total Horas:	**293**

Tabla 1 Tiempo empleado en el desarrollo del proyecto

A continuación, se muestra un gráfico de sectores para establecer una comparativa porcentual del tiempo empleado en cada categoría de tareas.

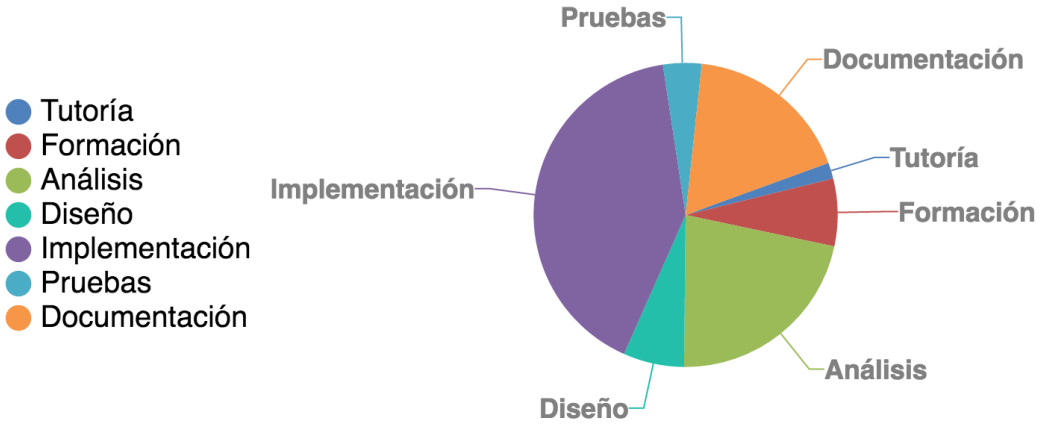


Ilustración 21 - Diagrama de sectores del tiempo empleado

3.6 Presupuesto

3.6.1 Factores de amortización

Se utilizan factores de amortización para calcular la cantidad que se le ha de cobrar al cliente por la utilización de elementos reutilizables para otros proyectos, o que no pierdan su valor después de terminar el mismo.

Elementos Software

Estimamos que un producto software puede tener una vida útil de aproximadamente 2 años, dada la frecuencia en los cambios de versión en los programas por parte de las compañías desarrolladoras.

Variables de entrada

- Duración del proyecto en días: 182.
- 2 años en días: 730.

Cálculo del factor de amortización

$$\frac{182}{730} = 0.2493 \simeq 25\%$$

Variable de salida

Porcentaje a cobrar en un elemento software reutilizable: 25%

Elementos Hardware

A diferencia de los elementos software, estimamos que los elementos hardware pueden tener una vida útil de 5 años.

Variables de entrada

- Duración del proyecto en días: 182.
- 5 años en días: 1826.

Cálculo del factor de amortización

$$\frac{182}{1826} = 0.099 \simeq 10\%$$

Variable de salida

Porcentaje a cobrar en un elemento hardware reutilizable: 10%

3.6.2 Elementos de valor añadido

Entendemos como elementos de valor añadido en el proyecto, aquellos que se subcontratan o se adquieren para el proyecto y que no son reutilizables después de la finalización del mismo.

A estos elementos, dada la inversión inicial por parte de la empresa o desarrollador autónomo, hay que añadirles un valor extra, generalmente variable en función del coste del servicio.

Para este proyecto, se ha utilizado un valor añadido del **7%** para todos los elementos necesarios.

3.6.3 Presupuesto de Costes

A continuación se detalla el presupuesto de costes del proyecto. Este presupuesto es el que utiliza la empresa o particular proveedor del servicio, y se utiliza posteriormente para calcular el presupuesto para el cliente.

Item	Subitem	Concepto	Cantidad	Precio Unitario	F.A / V.A	TOTAL
1		**Desarrollo de la aplicación**				
	001	Análisis	62	50,00 €		3.100,00 €
	002	Diseño	34	45,00 €		1.530,00 €
	003	Implementación y pruebas	257	35,00 €		8.995,00 €
	004	Documentación	63	25,00 €		1.575,00 €
2		**Elementos software**				
	001	Microsoft Office Project 2010	1	615,00 €	28%	172,20 €
	002	Enterprise Architect 9	1	164,00 €	28%	45,92 €
3		**Elementos hardware**				
	001	Dispositivo móvil Android	1	350,00 €	11%	38,50 €
	002	Dispositivo móvil iOS	1	600,00€	11%	66,00 €
	003	Ordenador portátil	1	950,00 €	11%	104,50 €
4		**Otros elementos de valor añadido**				
	001	Gastos de oficina (papel, impresión, etc.)	1	75,00 €	7%	80,25 €
	002	LucidChart Premium (Subscripción)	3	9,95 €	7%	31,94 €
	003	Mantenimiento de servidor de pruebas	4	45,00 €	7%	192,60 €
					Subtotal aplicación	15.200,00 €
					Subtotal elementos amortizables o de valor añadido	731,91 €
					Subtotal	15.931,91 €
					Beneficio industrial (18 %)	2.867,75 €
					Total	18.799,66 €

3.6.4 Presupuesto para el Cliente

Partiendo del presupuesto de costes mostrado en la página anterior, se genera el presupuesto para el cliente.

En este presupuesto es de vital importancia ocultar los elementos utilizados durante el proyecto que no se incluyan en el producto final, ya que no son elementos que vayamos a entregar al cliente, sino los medios con los que conseguimos desarrollar nuestro producto.

Refiriéndonos a este proyecto, esto quiere decir que debemos ocultar los elementos software y hardware empleados, así como los de valor añadido. Conseguimos esto prorrateando estos gastos entre los productos finales, en este caso, la aplicación Savour.

Para ello, ya que tenemos dividido el desarrollo de la aplicación en varios subitems, calculamos el porcentaje de participación de cada uno de ellos dividiendo su precio bruto individual entre el subtotal bruto de la aplicación.

Concepto	**Participación**
Análisis	14,90%
Diseño	8,17%
Implementación	48,08%
Pruebas	13,70%
Documentación	15,15%

El coste para el cliente en cada uno de los subitems, será el coste total del presupuesto de costes por el porcentaje de participación de cada uno de ellos.

Finalmente, es necesario incluir los impuestos de venta correspondientes, en este caso el I.V.A.

Item	**SubItem**	**Concepto**	**Cantidad**	**Precio Unitario**
1		**Aplicación Savour**	1	
	001	Análisis		2.801,15 €
	002	Diseño		1.535,93 €
	003	Implementación		9.038,88 €
	004	Pruebas		2.575,55 €
	005	Documentación		2.848,15 €
			Subtotal	18.799,66 €
			IVA (21%)	3.947,93 €
			Total	22.747,58€

Capítulo 4. Análisis

4.1 Identificación de los Subsistemas en la Fase de Análisis

En el caso concreto de este proyecto, por tratarse de una aplicación web adaptable a dispositivos móviles, veo conveniente separar el análisis de la aplicación en 2 subsistemas bien diferenciados.

4.1.1 Descripción de los Subsistemas

Subsistema aplicación web adaptable para el negocio

Se trata de la aplicación web base que contendrá el núcleo funcional y que podrá accederse desde cualquier dispositivo que disponga de un navegador web. Además, contará con aplicaciones móviles adaptadas para Android, iOS y Windows Phone.

Será utilizada de forma exclusiva por el negocio, debiendo proporcionar credenciales de acceso a la plataforma. Esta aplicación será el eje principal del proyecto, sobre la que se construirán el resto de elementos.

Subsistema aplicación móvil para el cliente

Los clientes podrán comunicarse con el backend de la aplicación web del negocio utilizando una aplicación móvil nativa.

4.2 Requisitos del Sistema

4.2.1 Obtención de los Requisitos del Sistema

4.2.1.1 Requisitos del subsistema aplicación web adaptable para el negocio

Requisitos Funcionales

Código	Nombre Requisito	Descripción del Requisito
RF 1	Autentificación	El sistema debe disponer de un mecanismo de autentificación de usuarios.
RF 1.1	Autentificación mediante usuario y contraseña	Acceso al sistema mediante usuario y contraseña.
RF 1.2	Logout	Se podrá salir del sistema de forma manual o automáticamente tras un tiempo de inactividad.
RF 2	Selección de vistas	En función del rol de usuario, podrán seleccionarse ciertas vistas de la aplicación.
RF 3	Vista de barra	Se podrá ver un listado de las bebidas solicitadas ordenadas por prioridad.
RF 3.1	Marca de bebida como preparada	El camarero responsable de las bebidas podrá marcarla como preparada.
RF 4	Vista de sala	Listado de todas las mesas del local con su estado.
RF 4.1	Creación de comanda	Creación de una nueva comanda asociada a una mesa.
RF 4.2	Modificación de comanda	Modificación de una comanda existente.
RF 4.3	Eliminación de comanda	Eliminación de una comanda existente.
RF 4.4	Marca de pasos de mesa	Actualizar el estado de una mesa en función de los servidos.
RF 4.5	Generación de facturas	Generar factura asociada a una comanda.
RF 4.6	Marca de factura como pagada.	Finalizar la comanda y liberar la mesa una vez que haya sido pagada.
RF 5	Vista de cocina	Listado de platos ordenados por prioridad y respetando el tiempo de los pasos.
RF 5.1	Marca de plato como preparado.	El responsable de cocina podrá marcar un plato como preparado.
RF 6	Vista de administración	Menú de administración con opciones para la administración de usuarios, CRM, estadísticas y gestión de la carta.
RF 6.1	Gestión de carta	Definición de los productos del negocio.
RF 6.1.1	Creación de grupos de producto	Definir un nuevo grupo de productos.
RF 6.1.2	Modificación de grupos de producto	Modificación de un grupo de producto existente.
RF 6.1.3	Eliminación de grupos de producto	Eliminar grupo de producto existente.
RF 6.1.4	Creación de producto	Creación de un nuevo producto.
RF 6.1.5	Modificación de producto	Modificación de un producto existente.
RF 6.1.6	Eliminación de producto	Eliminación de un producto existente.

RF 6.1.7	Marca de producto como no disponible.	Ocultar un producto sin necesidad de eliminarlo.
RF 6.2	Administración de usuarios	Acceso a pantalla de administración de usuarios y roles
RF 6.2.1	Creación de usuario	Creación de un nuevo usuario.
RF 6.2.2	Modificación de usuario	Modificación de un usuario existente.
RF 6.2.3	Eliminación de usuario	Eliminación de un usuario existente.
RF 6.2.4	Desactivación de usuario	Desactivar un usuario sin necesidad de eliminarlo.
RF 6.3	Pantalla de estadísticas.	Acceso a dashboard de estadísitcas.
RF 6.3.1	Estadísticas de ocupación	Estadisticas de ocupación del local con filtros de fechas.
RF 6.3.2	Estadísticas de platos más solicitados	Listado de los platos más solicitados con filtros de fechas.
RF 6.3.3	Estadísticas de Horarios populares	Estadísticas de los horarios con mayor ocupación del local con filtros de fechas.
RF 6.3.4	Estadísticas de facturación	Estadísticas de facturación con filtros de fechas.
RF 6.4	Menú CRM	Acceso a pantalla para la relación de fidelización con los clientes.
RF 6.4.1	Valoración de usuarios	Ver las valoraciones obtenidas con filtros de fechas.
RF 6.4.2	Respuesta a comentarios	Responder a los comentarios de los clientes.
RF 6.4.3	Publicación de Newsletters	Envío de notificaciones de promoción y fidelización.

Requisitos no funcionales

Código	Nombre Requisito	Descripción del Requisito
RNF 1	Adaptabilidad a dispositivos móviles	El diseño web debe ser perfectamente adaptable a dispositivos móviles, con un ancho mínimo de 480px.
RNF 2	Modo offline	La aplicación web del local debe seguir funcionando incluso si no hay conectividad a internet, sincronizando los datos en la nube al reestablecerse la conexión.
RNF 3	Seguridad en los datos	El envío de datos a través de la red debe utilizar algún mecanismo de encriptación.
RNF 4	Escalabilidad	El sistema debe estar preparado desde el principio para poder escalar horizontalmente.

4.2.1.2 Requisitos del subsistema aplicación móvil para el cliente

Requisitos Funcionales

Código	Nombre Requisito	Descripción del Requisito
RF 7	Identificación de local y mesa	Identificación manual o automática utilizando un código QR o NFC.
RF 8	Creación de comanda	Creación de una nueva comanda
RF 9	Asistencia	Solicitar la asistencia de un camarero.
RF 10	Pantalla de pago	Acceso a menú de pagos.
RF 10.1	Pago efectivo	Solicitar que un responsable entregue una factura y realice el cobro.
RF 10.2	Pago virtual	Pago mediante tarjeta de crédito o paypal.
RF 11	Valoraciones	Será posible realizar una valoración de entre 1 y 5 estrellas al finalizar un pago y un comentario opcional.

Requisitos no funcionales

Código	Nombre Requisito	Descripción del Requisito
RFN 5	Compatibilidad	Sistema operativo Ios 8 o superior.

4.2.2 Identificación de Actores del Sistema

4.2.2.1 Identificación de Actores en el subsistema aplicación web base

Administrador

Para cada negocio que solicitase adherirse a Savour, se entregaría un usuario administrador. Este usuario dispone de todos los permisos dentro del sistema, incluso de la creación de nuevos usuarios administradores. La idea es que sea el gerente del local o encargado de la empresa el utilice este tipo de usuario.

Empleado

Los trabajadores del local en cambio no tendrán acceso a estadísticas de facturación, al submenú CRM o a la propia gestión de productos. Por ello, se plantea necesaria la creación del rol "Empleado". Este perfil podría incluso subdividirse en responsables de mesa, barra o cocina, que tuviesen acceso exclusivo a las respectivas vistas.

4.2.2.2 Identificación de Actores en el subsistema aplicación móvil para el cliente

A diferencia del subsistema aplicación web base, para el subsistema aplicaciones móviles híbridas se identifica un único actor.

Cliente

Las aplicaciones móviles estarán destinadas exclusivamente a los clientes del local. El personal de mesa también podría hacer uso de dispositivos móviles para realizar las comandas, pero en este caso se utilizaría la adaptación móvil de la aplicación web para el negocio.

El principal motivo es que los clientes del local necesitarían una aplicación nativa para ciertas características como la identificación de la mesa, notificaciones push, etc.

4.2.3 Especificación de Casos de Uso

4.2.3.1 Especificación de Casos de Uso en el subsistema aplicación web base para el negocio

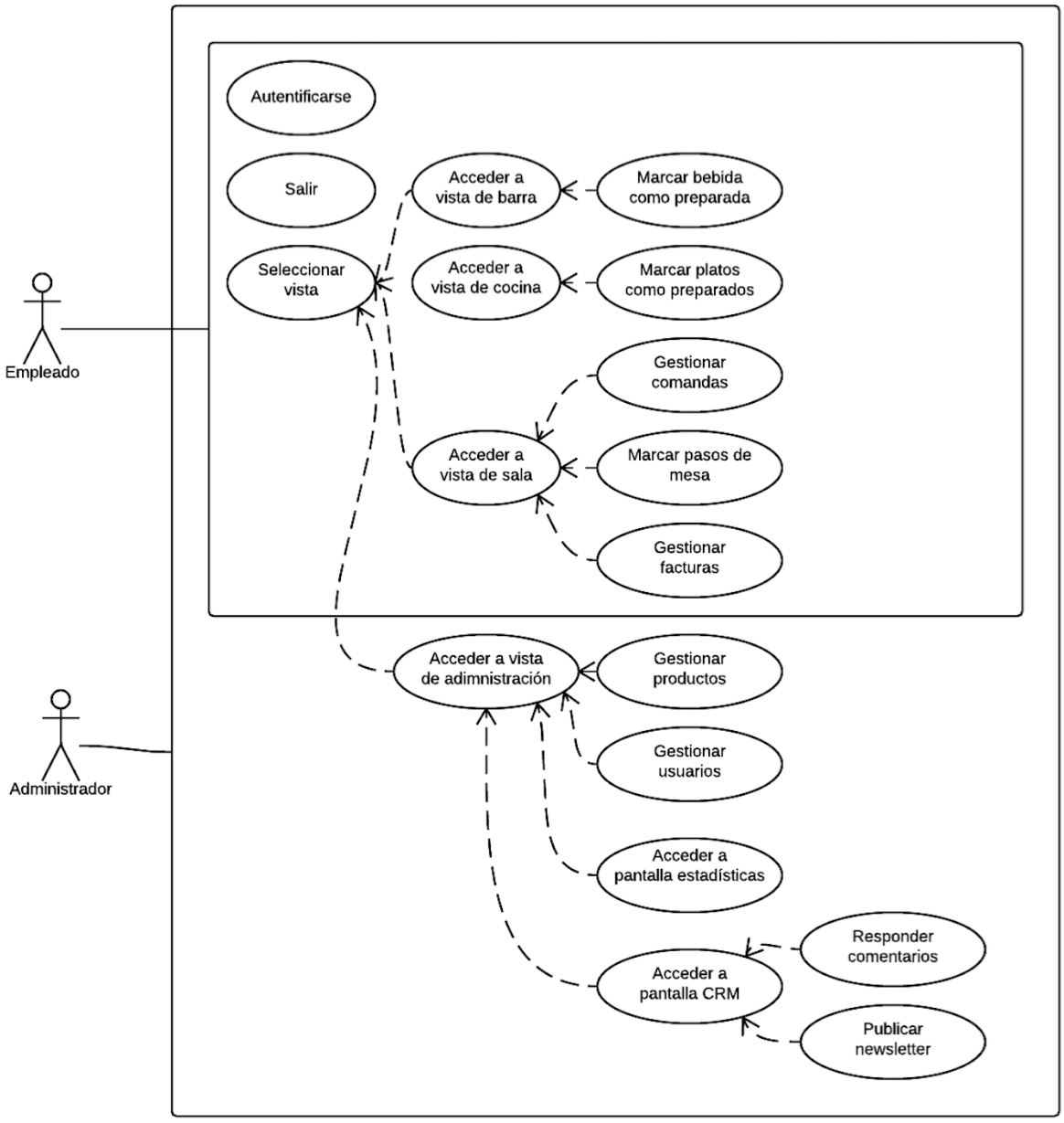


Ilustración 22 - Diagrama de casos de uso del subsistema aplicación web base

CU1: Autentificarse

Caso de Uso 1: Autentificarse	
Descripción:	Permite al usuario autentificarse en la aplicación.
Precondición:	El usuario no debe estar autentificado.
Escenario principal:	1. El usuario accede a la pantalla principal. 2. El usuario introduce usuario y contraseña. 3. El usuario envía el formulario.
Flujo alternativo:	1.1 El sistema no dispone de conexión de red y el envío del formulario produce un error.
Flujo alternativo:	-

CU2: Salir

Caso de Uso 2: Salir	
Descripción:	Permite al usuario salir de la aplicación.
Precondición:	El usuario debe estar autentificado.
Escenario principal:	1. En cualquier pantalla, el usuario presina el botón de “Salir” en la esquina superior derecha. 2. El sistema redirige al usuario a la pantalla inicial de autentificación.
Flujo alternativo:	1.1 El usuario cierra el navegador directamente y la sesión expira pasado un pequeño periodo de tiempo.
Flujo alternativo:	-

CU3: Seleccionar vista

Caso de Uso 3: Seleccionar vista	
Descripción:	Permite al usuario seleccionar una de las 4 vistas disponibles.
Precondición:	El usuario debe estar autentificado.
Escenario principal:	1. En cualquier pantalla, el usuario puede presionar el botón de selección de vistas. 2. El usuario presiona una de las 4 vistas en la pantalla de selección. 3. La nueva vista se carga y aparece en pantalla.
Flujo alternativo:	1.1 Cuando el usuario se autentifica por primera vez, aparece por defecto la pantalla de selección de vista.
Flujo alternativo:	-

CU4: Acceder a vista de barra

Caso de Uso 4: Acceder a vista de barra	
Descripción:	Acceso a la vista de barra con el listado de bebidas a servir para cada comanda.
Precondición:	El usuario debe estar autentificado.
Escenario principal:	1. En cualquier pantalla, el usuario puede presionar el botón de selección de vistas. 2. El usuario presiona el botón de vista de barra en la pantalla de selección de vistas. 3. La nueva vista se carga y aparece en pantalla.
Flujo alternativo:	-
Flujo alternativo:	-

CU5: Marcar bebida como preparada

Caso de Uso 5: Marcar bebida como preparada	
Descripción:	Una vez preparada la bebida se podrá marcar como preparada para que el personal de mesa se la entregue a los clientes.
Precondición:	El usuario debe estar autentificado.
Escenario principal:	1. En cualquier pantalla, el usuario puede presionar el botón de selección de vistas. 2. El usuario presiona el botón de vista de barra en la pantalla de selección de vistas. 3. La nueva vista se carga y aparece en pantalla. 4. El usuario hace click sobre el icono “preparado” de la comanda concreta. 5. La vista se actualiza y la comanda de bebida desaparece.
Flujo alternativo:	-
Flujo alternativo:	-

CU6: Acceder a vista de sala

Caso de Uso 6: Acceder a vista de sala	
Descripción:	Acceso a la vista de sala con el listado de mesas y sus estados.
Precondición:	El usuario debe estar autentificado.
Escenario principal:	1. En cualquier pantalla, el usuario puede presionar el botón de selección de vistas. 2. El usuario presiona el botón de vista de sala en la pantalla de selección de vistas.

	3. La nueva vista se carga y aparece en pantalla. 4. Sobre la nueva vista el usuario puede acceder a una pantalla de detalle para cada mesa en cuestión.
Flujo alternativo:	-
Flujo alternativo:	-

CU7: Gestionar comanda

Caso de Uso 7: Gestionar comanda	
Descripción:	Crear, modificar, o anular una comanda.
Precondición:	El usuario debe estar autentificado.
Escenario principal:	1. En cualquier pantalla, el usuario puede presionar el botón de selección de vistas. 2. El usuario presiona el botón de vista de mesa en la pantalla de selección de vistas. 3. La nueva vista se carga y aparece en pantalla. 4. El usuario puede crear una comanda utilizando el botón correspondiente, para lo que tendría que rellenar los datos necesarios como número de mesa, comensales, listado de comida y bebida, etc.
Flujo alternativo:	4.1 El usuario puede modificar una comanda existente añadiendo o eliminando ítems, o simplemente actualizando los datos de cabecera de la comanda.
Flujo alternativo:	4.2 El usuario puede finalizar una comanda, eliminarla directamente o realizar una devolución.

CU8: Marcar paso de mesa

Caso de Uso 8: Marcar paso de mesa	
Descripción:	El personal de mesa será el encargado de marcar los pasos de mesa, según los clientes vayan finalizando los platos.
Precondición:	El usuario debe estar autentificado.
Escenario principal:	1. En cualquier pantalla, el usuario puede presionar el botón de selección de vistas. 2. El usuario presiona el botón de vista de mesa en la pantalla de selección de vistas. 3. La nueva vista se carga y aparece en pantalla. 4. El usuario selecciona la mesa sobre la que quiere marcar el paso.

	5. Sobre la pantalla de detalla de la mesa, el usuario utiliza el botón de marcar paso, sobre el paso de mesa correspondiente. 6. La vista de cocina recibe el aviso y se actualiza en concordancia.
Flujo alternativo:	5.1 El usuario selecciona el último paso de mesa, y la comanda pasa a el estado "finalizado por el cliente".
Flujo alternativo:	-

CU9: Gestionar facturas

Caso de Uso 9: Gestionar facturas	
Descripción:	Permite generar una factura al cliente y procesar el cobro.
Precondición:	El usuario debe estar autentificado. El estado por el cual es posible generar una factura para una mesa debe ser: "finalizado por el cliente".
Escenario principal:	1. En cualquier pantalla, el usuario puede presionar el botón de selección de vistas. 2. El usuario presiona el botón de vista de mesa en la pantalla de selección de vistas. 3. La nueva vista se carga y aparece en pantalla. 4. El usuario selecciona la mesa sobre la que quiere generar una factura. 5. Sobre la vista de detalle se selecciona "generar factura". 6. Se genera una factura que posteriormente se puede imprimir y se actualiza el estado a "en espera de pago de cliente". 7. Una vez recibido el pago, se procederá a marcar la factura como pagada. 8. La comanda se archivará y se actualizará la vista de mesas en consecuencia liberando la mesa facturada.
Flujo alternativo:	- 6.1 En el caso de algún error en la factura, es posible realizar cualquier modificación siempre y cuando aún no se haya marcado como pagada, mediante el botón "modificar factura".
Flujo alternativo:	-

CU10: Acceder a vista de cocina

Caso de Uso 10: Acceder a vista de cocina	
Descripción:	Acceso a la vista de cocina con el listado de platos a servir para cada comanda.
Precondición:	El usuario debe estar autentificado.

Escenario principal:	1. En cualquier pantalla, el usuario puede presionar el botón de selección de vistas. 2. El usuario presiona el botón de vista de cocina en la pantalla de selección de vistas. 3. La nueva vista se carga y aparece en pantalla.
Flujo alternativo:	-
Flujo alternativo:	-

CU11: Marcar platos como preparados

Caso de Uso 11: Marcar platos como preparados	
Descripción:	El personal de cocina deberá marcar los platos como preparados para notificar al personal de mesa que están listos para ser recogidos.
Precondición:	El usuario debe estar autentificado.
Escenario principal:	1. En cualquier pantalla, el usuario puede presionar el botón de selección de vistas. 2. El usuario presiona el botón de vista de cocina en la pantalla de selección de vistas. 3. La nueva vista se carga y aparece en pantalla. 4. Sobre la comanda concreta, podrán seleccionarse los platos preparados de forma individual. 5. La vista de mesa se actualizará mostrando un aviso para la mesa en concreto.
Flujo alternativo:	-
Flujo alternativo:	-

CU12: Acceder a vista de administración

Caso de Uso 12: Acceder a vista de administración	
Descripción:	Los usuarios administradores tienen acceso especial a esta vista, donde pueden gestionar los productos (carta), revisar estadísiticas y/o controlar la fidelización de los clientes.
Precondición:	El usuario debe estar autentificado. El rol del usuario debe ser administrador.
Escenario principal:	1. En cualquier pantalla, el usuario puede presionar el botón de selección de vistas. 2. El usuario presiona el botón de vista de administración en la pantalla de selección de vistas. 3. La nueva vista se carga y aparece en pantalla.
Flujo alternativo:	-
Flujo alternativo:	-

CU13: Gestionar productos (carta)

Caso de Uso 13: Gestionar productos	
Descripción:	Los usuarios administradores podrán añadir, modificar, ocultar o eliminar elementos de la carta.
Precondición:	El usuario debe estar autentificado. El usuario debe tener el rol de administrador.
Escenario principal:	1. En cualquier pantalla, el usuario puede presionar el botón de selección de vistas. 2. El usuario presiona el botón de vista de administración en la pantalla de selección de vistas. 3. La nueva vista se carga y aparece en pantalla. 4. El usuario accede al menú de gestión de productos 5. En la nueva pantalla se pueden crear, modificar o eliminar grupos de productos (jerarquía de árbol) o añadir, modificar, ocultar o eliminar productos.
Flujo alternativo:	-
Flujo alternativo:	-

CU14: Gestionar usuarios

Caso de Uso 14: Gestionar usuarios	
Descripción:	Los usuarios administradores pueden añaidr, modificar, desactivar o eliminar usuarios.
Precondición:	El usuario debe estar autentificado. El usuario debe tener el rol de administrador.
Escenario principal:	1. En cualquier pantalla, el usuario puede presionar el botón de selección de vistas. 2. El usuario presiona el botón de vista de administración en la pantalla de selección de vistas. 3. La nueva vista se carga y aparece en pantalla. 4. El usuario accede al menú de gestión de usuarios 5. En la nueva pantalla se pueden crear, modificar, desactivar o eliminar usuarios.
Flujo alternativo:	-
Flujo alternativo:	-

CU15: Acceder a pantalla de estadísticas

Caso de Uso 15: Acceder a la pantalla de estadísticas	
Descripción:	Los usuarios administradores tienen acceso a un dashboard de estadísitcas variadas sobre el negocio.
Precondición:	El usuario debe estar autentificado. El usuario debe tener el rol de administrador.
Escenario principal:	1. En cualquier pantalla, el usuario puede presionar el botón de selección de vistas. 2. El usuario presiona el botón de vista de administración en la pantalla de selección de vistas. 3. La nueva vista se carga y aparece en pantalla. 4. El usuario accede al menú de estadísticar. 5. Para cada una de las estadísitcas se pueden utilizar diferentes filtros.
Flujo alternativo:	-
Flujo alternativo:	-

CU16: Acceder a pantalla de CRM

Caso de Uso 16: Acceder a la pantalla de CRM	
Descripción:	Los usuarios administradores pueden acceder a una pantalla para el control de la fidelización de los clientes en la que puede leer y responder comentarios, revisar las valoraciones o publicar newsletters.
Precondición:	El usuario debe estar autentificado. El usuario debe tener el rol de administrador.
Escenario principal:	1. En cualquier pantalla, el usuario puede presionar el botón de selección de vistas. 2. El usuario presiona el botón de vista de administración en la pantalla de selección de vistas. 3. La nueva vista se carga y aparece en pantalla. 4. El usuario selecciona el menú CRM.
Flujo alternativo:	-
Flujo alternativo:	-

CU17: Responder a comentarios

Caso de Uso 17: Responder a comentarios de clientes	
Descripción:	Los usuarios administradores pueden responder a comentarios de sus clientes desde el submenú CRM.
Precondición:	El usuario debe estar autentificado. El usuario debe tener el rol de adminsitrador.
Escenario principal:	1. En cualquier pantalla, el usuario puede presionar el botón de selección de vistas. 2. El usuario presiona el botón de vista de administración en la pantalla de selección de vistas. 3. La nueva vista se carga y aparece en pantalla. 4. El usuario selecciona el menú CRM 5. El usuario selecciona el submenú Comentarios de clientes. 6. Sobre la lista de comentarios, es posible reponder a cada uno de ellos, generando una notificación para el cliente concreto.
Flujo alternativo:	-
Flujo alternativo:	-

CU18: Publicar newsletter

Caso de Uso 18: Publicar newsletter	
Descripción:	Los usuarios administradores pueden enviar boletines de noticias periódicas a sus clientes.
Precondición:	El usuario debe estar autentificado. El usuario debe tener el rol de administrador.
Escenario principal:	1. En cualquier pantalla, el usuario puede presionar el botón de selección de vistas. 2. El usuario presiona el botón de vista de administración en la pantalla de selección de vistas. 3. La nueva vista se carga y aparece en pantalla. 4. El usuario selecciona el menú CRM. 5. El usuario selecciona el submenú Newsletters. 6. En la nueva pantalla es posible redactar un nuevo newsletter y enviarlo a los clientes en forma de notificación push.
Flujo alternativo:	-
Flujo alternativo:	-

Relación Casos de Uso – Requisitos en el subsistema aplicación web base

	CU 1	CU 2	CU 3	CU 4	CU 5	CU 6	CU 7	CU 8	CU 9	CU 10	CU 11	CU 12	CU 13	CU 14	CU 15	CU 16	CU 17	CU 18
RF 1	X	X																
RF 1.1	X																	
RF 1.2		X																
RF 2			X															
RF 3				X														
RF 3.1					X													
RF 4						X												
RF 4.1							X											
RF 4.2							X											
RF 4.3							X											
RF 4.4								X										
RF 4.5									X									
RF 4.6									X									
RF 5										X								
RF 5.1										X								
RF 6											X							
RF 6.1												X						
RF 6.1.1													X					
RF 6.1.2													X					
RF 6.1.3													X					
RF 6.1.4													X					
RF 6.1.5													X					
RF 6.1.6													X					
RF 6.1.7													X					
RF 6.2														X				

RF 6.2.1														X				
RF 6.2.2														X				
RF 6.2.3														X				
RF 6.2.4														X				
RF 6.3															X			
RF 6.3.1															X			
RF 6.3.2															X			
RF 6.3.3															X			
RF 6.3.4															X			
RF 6.4																X		
RF 6.4.1																	X	
RF 6.4.2																	X	
RF 6.4.3																		X

4.2.3.2 Especificación de casos de uso en el subsistema aplicación móvil

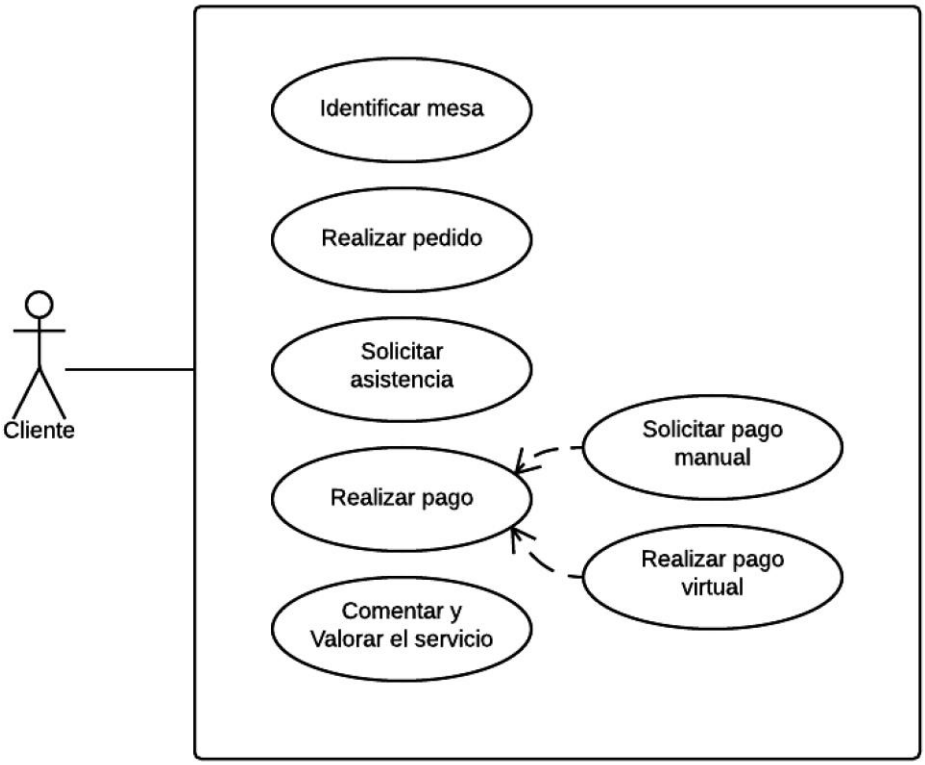


Ilustración 23 - Diagrama de casos de uso del subsistema aplicación móvil

CU 1: Identificar mesa y local

Caso de Uso 1: Identificar mesa y local	

Descripción:	Para poder realizar un nuevo pedido, el cliente deberá haber localizado previamente su mesa y el local. Para ello pueden darse los siguientes casos:
	• **Identificación manual**: El cliente selecciona el local a partir de un listado, o utiliza la aplicación móvil específica del local e introduce el número de mesa que deberá estar visible en el establecimiento.
	• **Identificación automática**:
	  ○ **Mediante código QR**: El cliente escanea un código QR visible en la mesa que la identifica junto con el local.
	  ○ **Mediante bluetooth/NFC**: El cliente acerca el dispositivo a una baliza sobre la mesa, que transmite mediante NFC o bluetooth el número de mesa y local.

Precondición:	-
Escenario principal:	1. El cliente accede a la aplicación

	2. Sobre una lista de locales el cliente selecciona aquel en el que se encuentra 3. El cliente introduce el número de mesa que se encontrará visible en algún lugar. 4. Aparece la pantalla de pedido.
Flujo alternativo1:	2.1. El cliente activa la cámara y escanea un código QR visible en algún lugar de la mesa. 3. Aparece la pantalla de pedido.
Flujo alternativo2:	2.2 El cliente acerca el dispositivo a una baliza visible sobre la mesa. 3. Aparece la pantalla de pedido

CU 2: Realizar pedido

Caso de Uso 2: Realizar pedido	
Descripción:	Una vez identificada la mesa y el local, el cliente podrá solicitar un nuevo pedido (comanda).
Precondición:	Haber identificado previamente la mesa y local.
Escenario principal:	1. El cliente accede a la pantalla de nuevo pedido 2. El cliente visualiza la carta virtual y añade productos al pedido. 3. Una vez finalizado el cliente selecciona “ordenar”. 4. El cliente recibe una confirmación.
Flujo alternativo:	-
Flujo alternativo:	-

CU 3: Solicitar asistencia

Caso de Uso 3: Solicitar asistencia	
Descripción:	El cliente podrá solicitar la atención de un camarero en cualquier momento.
Precondición:	Haber identificado previamente la mesa y el local.
Escenario principal:	1. El cliente pulsa sobre el icono de “solicitud de asistencia”.
Flujo alternativo:	-
Flujo alternativo:	-

CU 4: Realizar pago

Caso de Uso 4: Realizar pago	
Descripción:	El cliente podrá realizar el pago de un pedido en cualquier momento siempre que el pedido ya esté ordenado.
Precondición:	Haber identificado previamente la mesa y local. Haber realizado un pedido.
Escenario principal:	1. El cliente accede a su pedido.

	2. El cliente selecciona “pagar”. 3. El cliente accede a la pantalla de pago.
Flujo alternativo:	-
Flujo alternativo:	-

CU 5: Solicitar pago manual

Caso de Uso 5: Solicitar pago manual	
Descripción:	El cliente puede solicitar realizar el pago manual, mediante dinero en efectivo o TPV física.
Precondición:	Haber identificado previamente la mesa y local. Haber realizado un pedido. Haber solicitado el pago.
Escenario principal:	1. El cliente accede a la pantalla de pago. 2. El cliente selecciona la opción “Pago manual”. 3. El cliente selecciona “Efectivo o Tarjeta”. 4. El cliente espera al encargado de facturación para realizar el pago.
Flujo alternativo:	-
Flujo alternativo:	-

CU 6: Realizar pago virtual

Caso de Uso 6: Realizar pago virtual	
Descripción:	El cliente puede realizar el pago virtual a través de una pasarela de pago online.
Precondición:	Haber identificado previamente la mesa y local. Haber realizado un pedido. Haber solicitado el pago.
Escenario principal:	1. El cliente accede a la pantalla de pago. 2. El cliente selecciona la opción “Pago virtual”. 3. El cliente introduce los datos de su tarjeta en la nueva pantalla de pago y selecciona “pagar”. 4. El cliente recibe la confirmación del pago.
Flujo alternativo:	-
Flujo alternativo:	-

CU 7: Comentar y valorar el servicio

Caso de Uso 7: Comentar y valorar el servicio	
Descripción:	Una vez realizado el pago, el cliente podrá valorar el servicio y realizar comentarios.
Precondición:	Haber identificado previamente la mesa y local. Haber realizado un pedido. Haber realizado un pago. No haber valorado previamente el mismo pedido.
Escenario principal:	1. El cliente accede a sus últimos pedidos. 2. El cliente selecciona el pedido que desea valorar. 3. El cliente realiza un comentario y/o valoración (1-5 estrellas). 4. El cliente recibe una confirmación.
Flujo alternativo:	-
Flujo alternativo:	-

Relación Casos de Uso – Requisitos en el subsistema aplicación móvil

	CU 1: Id. mesa	CU 2: Realizar pedido	CU 3: Asistencia	CU 4: Realizar pago	CU 5: Pago manual	CU 6: Pago virtual	CU 7: Valorar y comentar
RF 7	X						
RF 8		X					
RF 9			X				
RF 10				X			
RF 10.1				X	X		
RF 10.2				X		X	
RF 11							X

4.3 Modelo de dominio

4.3.1 Diagramas del modelo del dominio

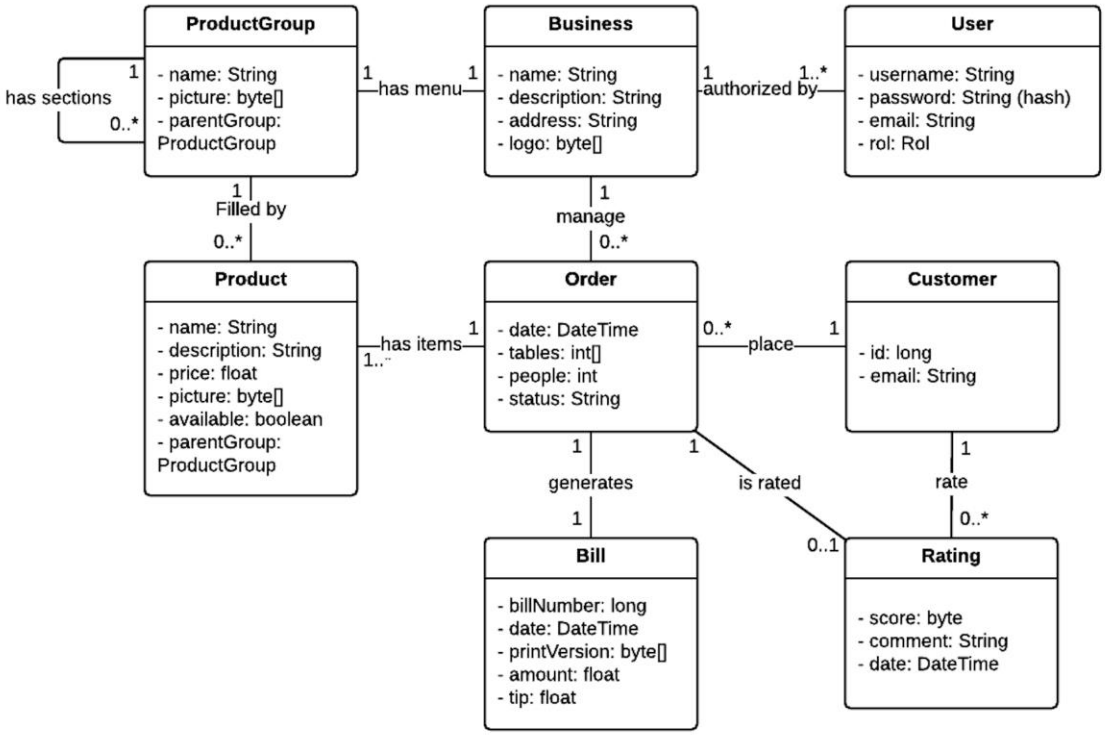


Ilustración 24 - Diagrama preliminar del modelo de dominio

4.3.2 Descripción de las Clases

Businesss

Como su nombre indica, representa un negocio de un bar o restaurante, y del que dependerán directa o indirectamente el resto de clases.

Cada negocio deberá tener un identificador único, nombre, descripción, dirección postal y su logo asociado.

El negocio además tiene un grupo de productos padre, que representa la raíz de la carta.

User

Identifica a un usuario dentro de la aplicación, que podrá autentificarse mediante usuario/email y contraseña.

ProductGroup

Representa una agrupación de productos, que viene a ser un nodo en el árbol de la carta virtual.

Product

Se trata de un producto del bar o restaurante, representado como una hoja en el árbol de la carta virtual. Podría ser cualquier tipo de comida o bebida que pudiese aparecer en la carta, del que debería codificarse su nombre, descripción, foto asociada, precio y disponibilidad.

Order

Esta clase representa el pedido de un cliente, lo que viene a ser una **comanda** en la hostelería tradicional. Es por ello que deberá representar fielmente su significado, recogiendo el listado de productos pedidos, la mesa o mesas utilizadas, fecha, número de comensales y estado.

Bill

Cobro asociado a un pedido. Debe disponer de un identificador único, fecha y una versión imprimible de solo lectura donde se plasmaría el contenido de su comanda asociada.

Customer

Los clientes que realicen sus pedidos desde la aplicación móvil vendrán representados mediante esta clase. Dispondrán de identificador único y opcionalmente un email asociado.

Rating

Se trata de una evaluación de un pedido por parte de un cliente. En dicha evaluación podrá otorgarse una puntuación entre 1 y 5 estrellas y/o realizar un comentario sobre el servicio.

4.4 Análisis de Interfaces de Usuario

4.4.1 Diagramas de navegabilidad

Diagrama de navegabilidad subsistema aplicación web adaptable para el negocio

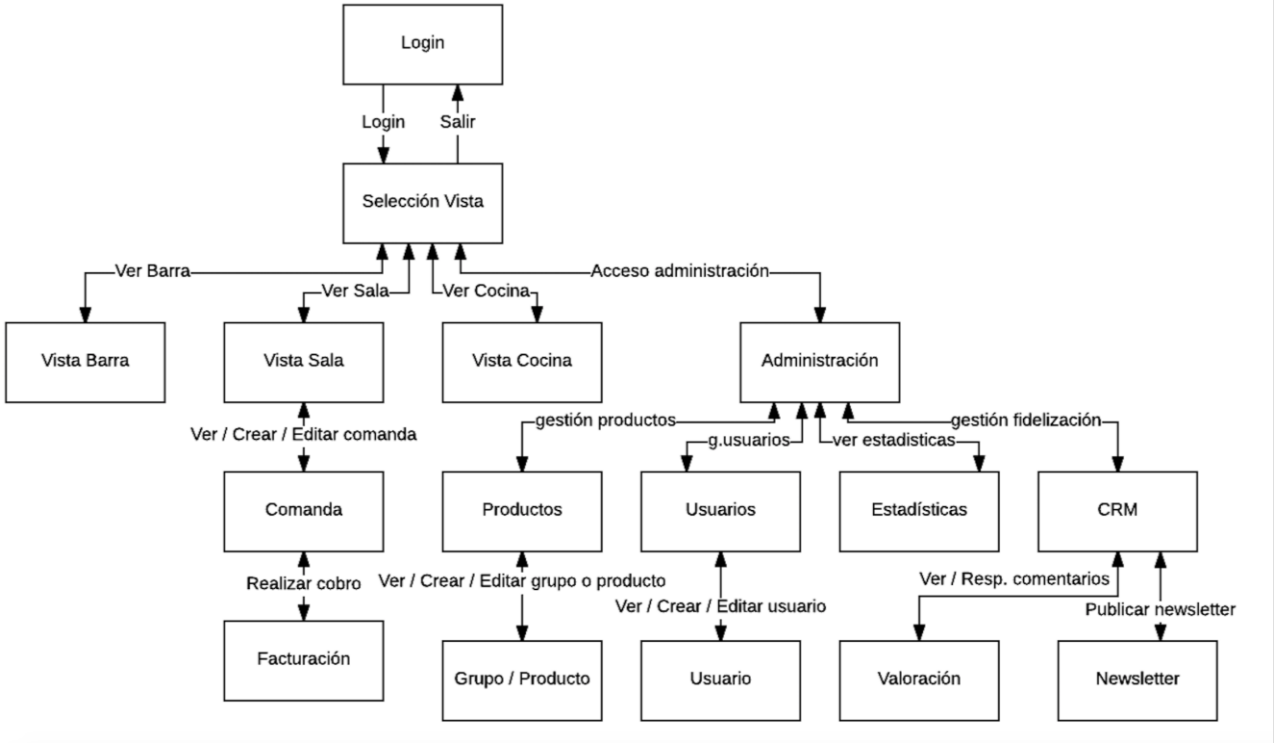


Ilustración 25 - Diagrama de navegavilidad subsistema aplicación web para el negocio

Diagrama de navegabilidad subsistema aplicación móvil para el cliente

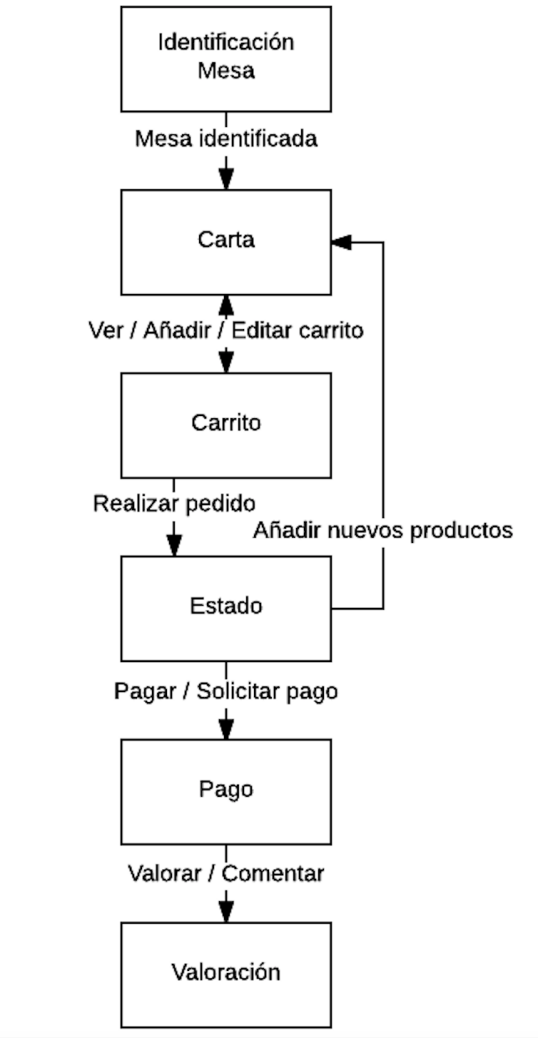
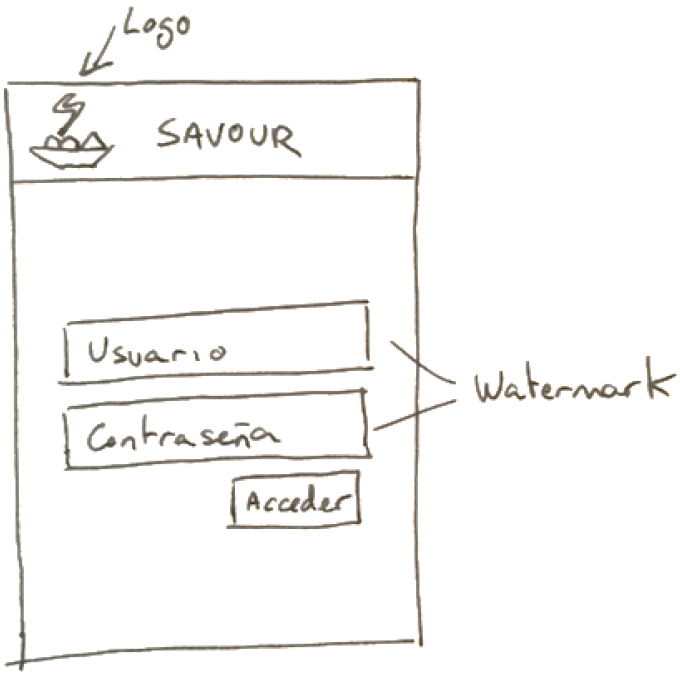


Ilustración 26 - Diagrama de navegabilidad subsistema aplicación móvil para el cliente

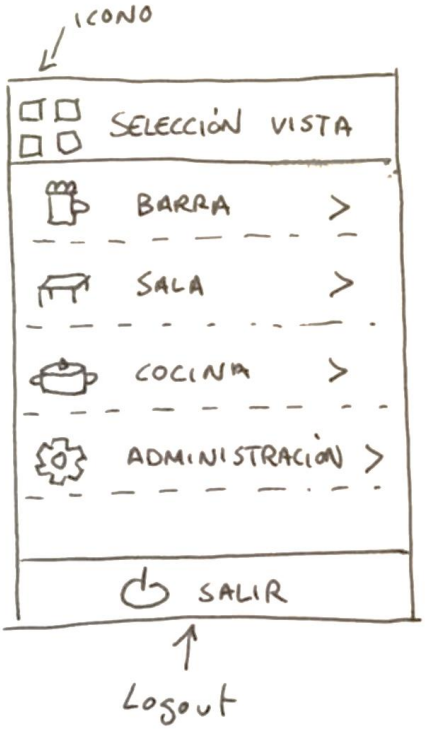
4.4.2 Prototipos de pantallas

4.4.2.1 Prototipos de pantallas en el subsistema aplicación web

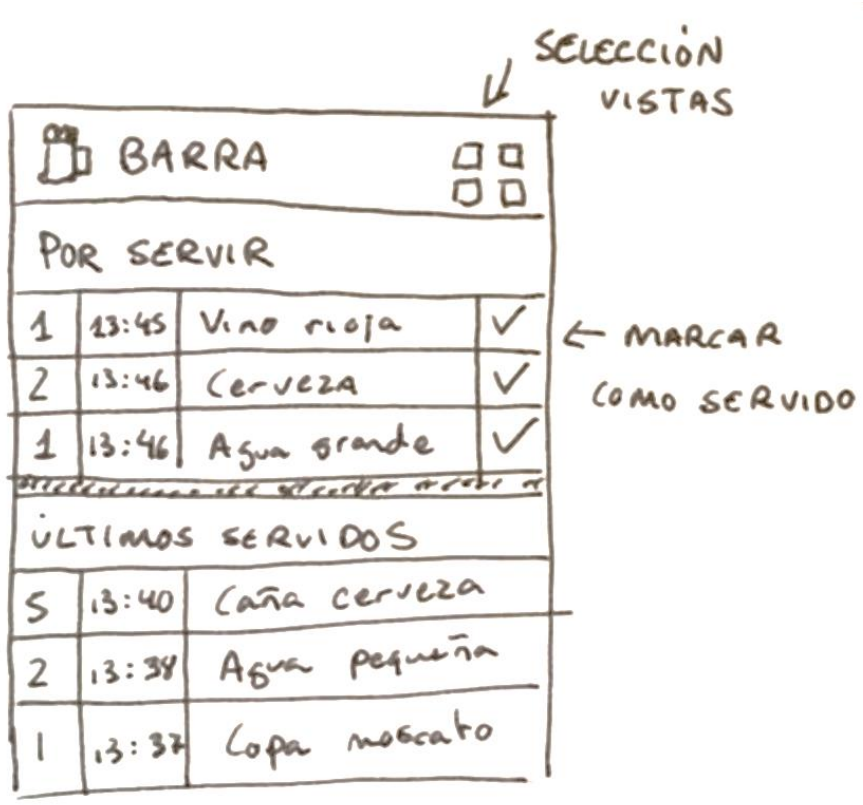
Login



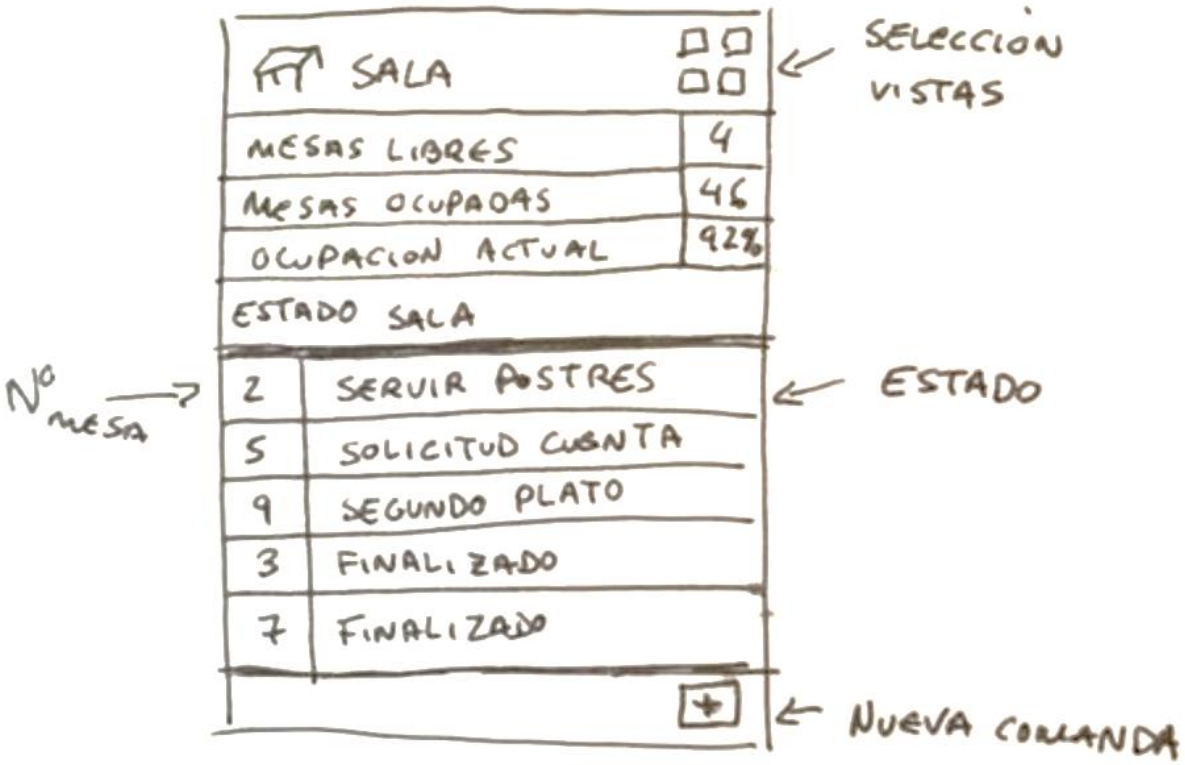
Selección Vista



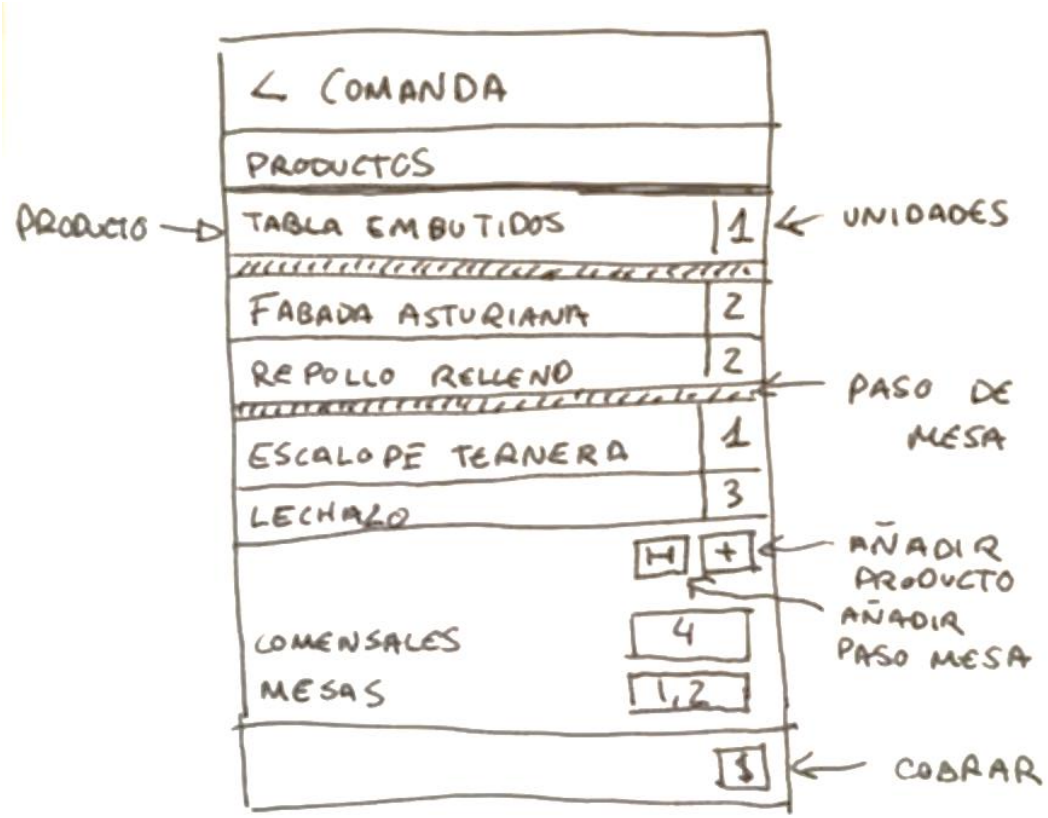
Vista de barra



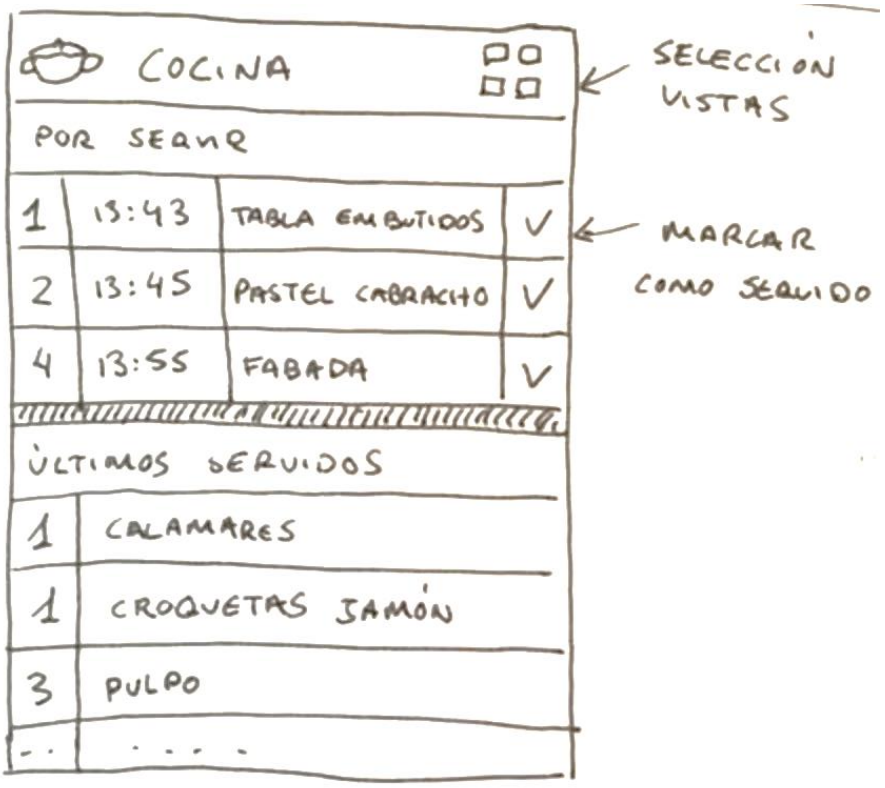
Vista de sala



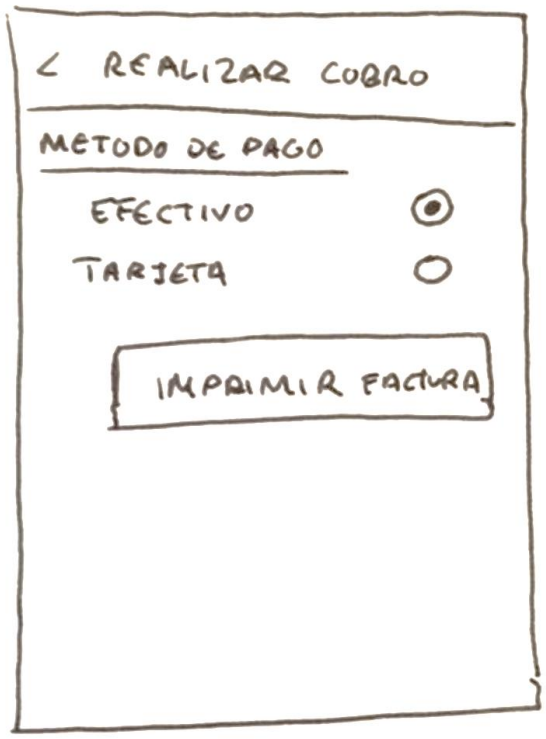
Comanda



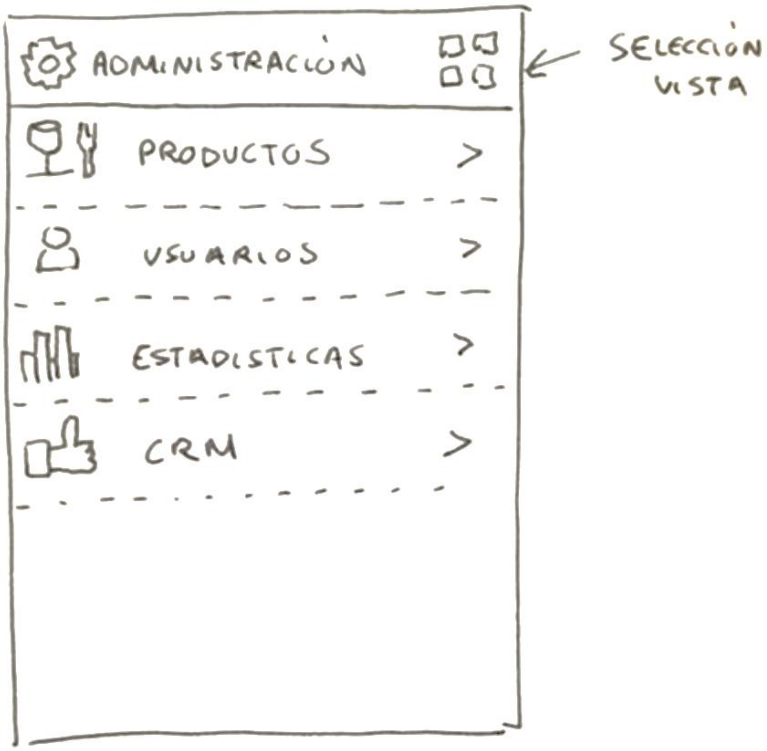
Vista de Cocina



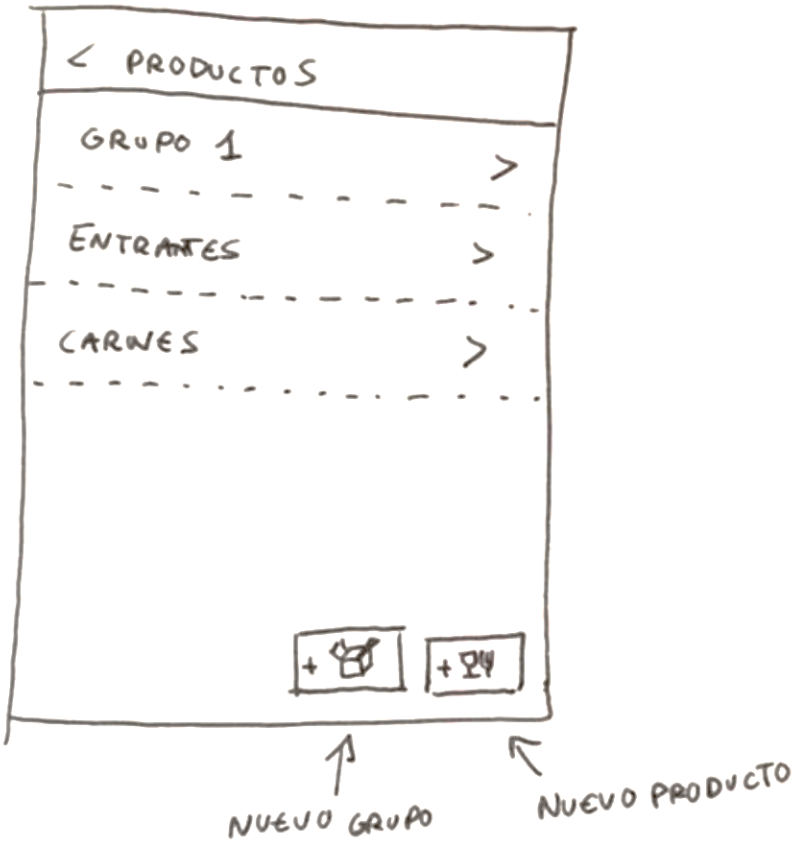
Realizar cobro



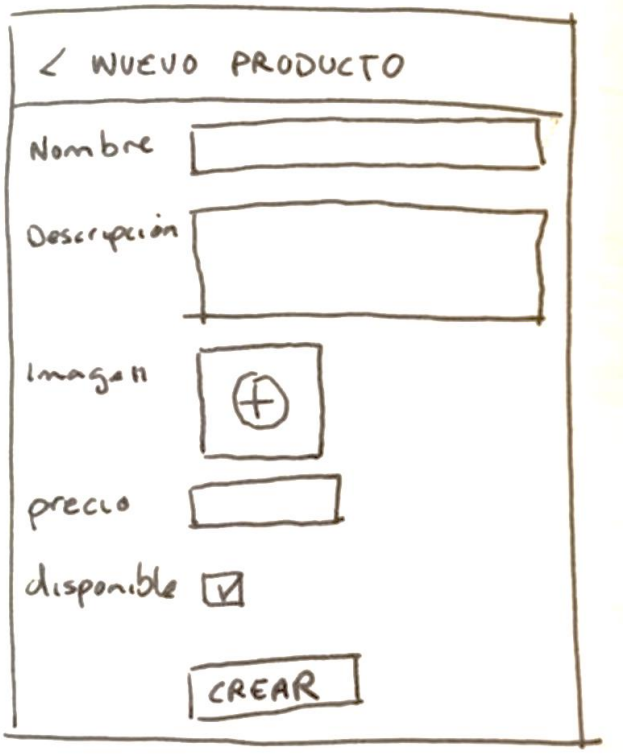
Vista de administración



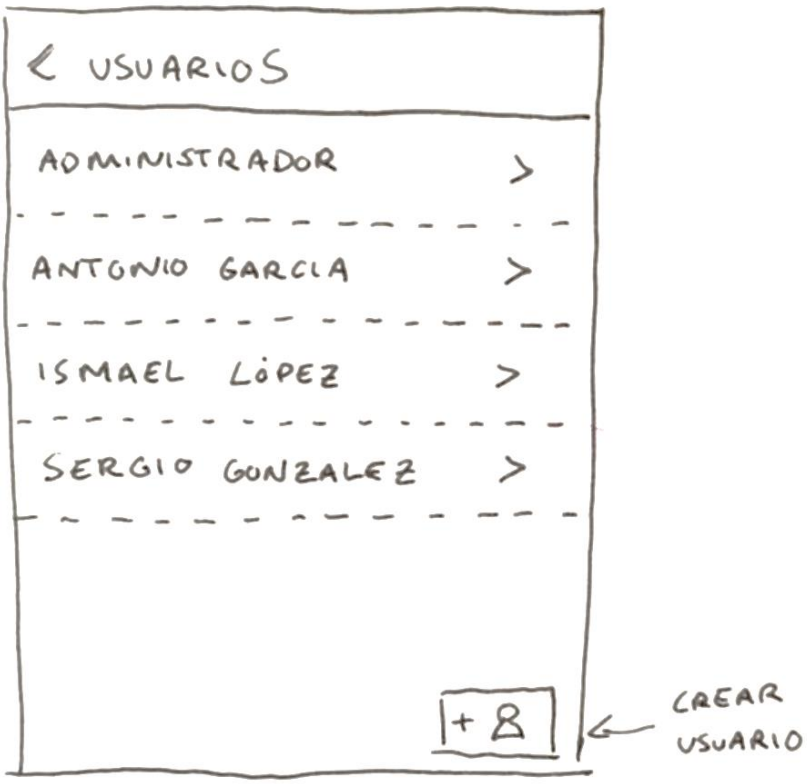
Productos



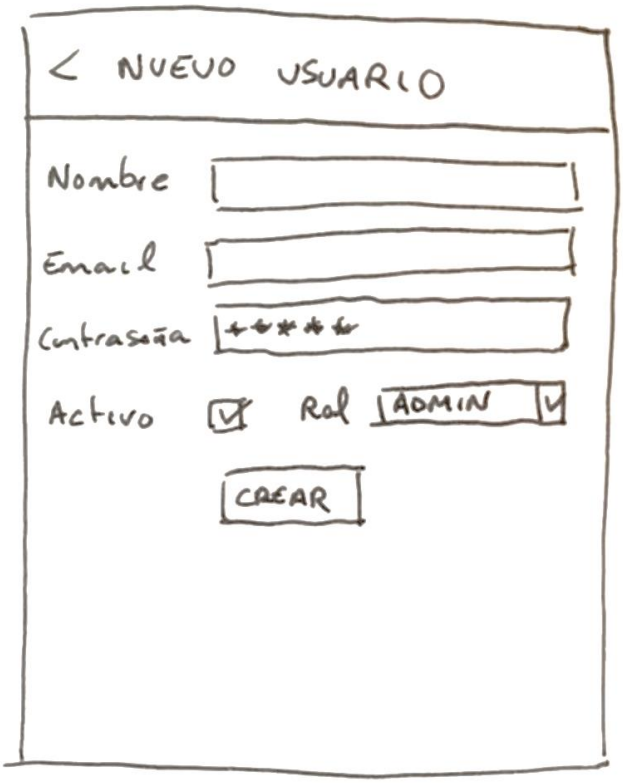
Detalle Producto



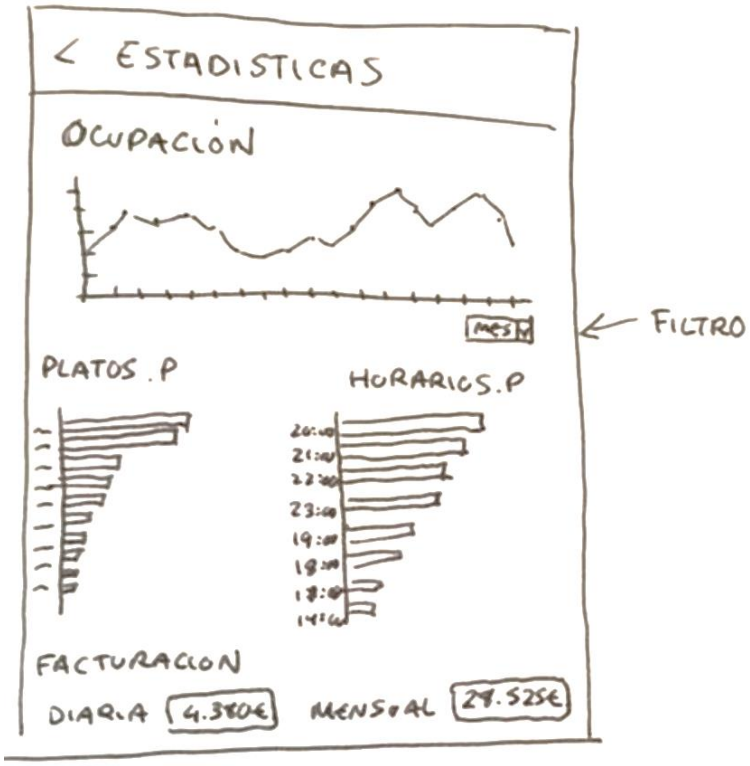
Usuarios



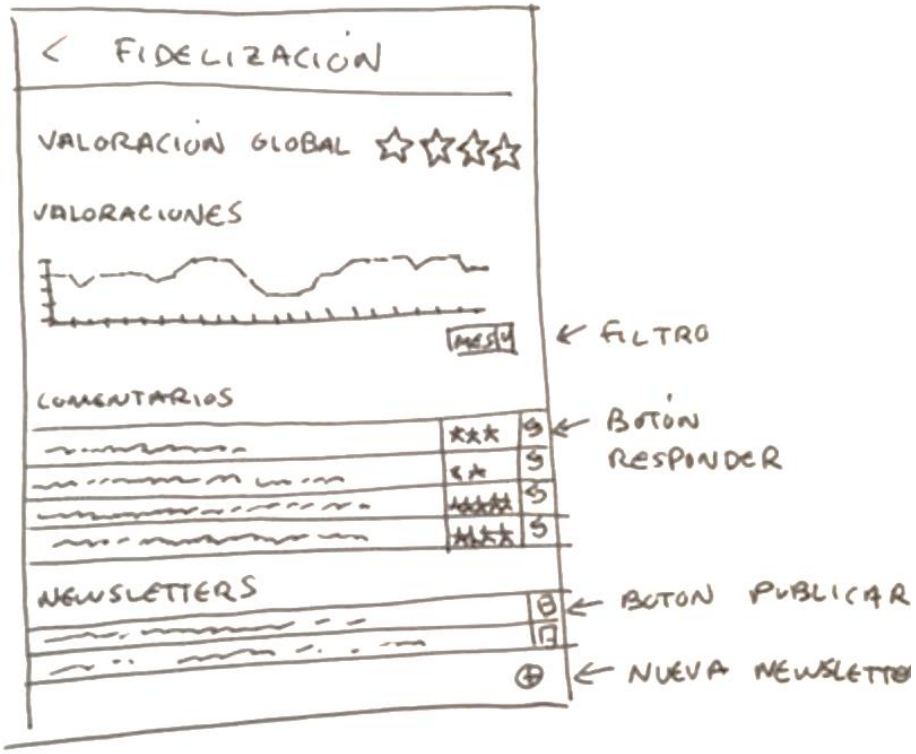
Detalle Usuario



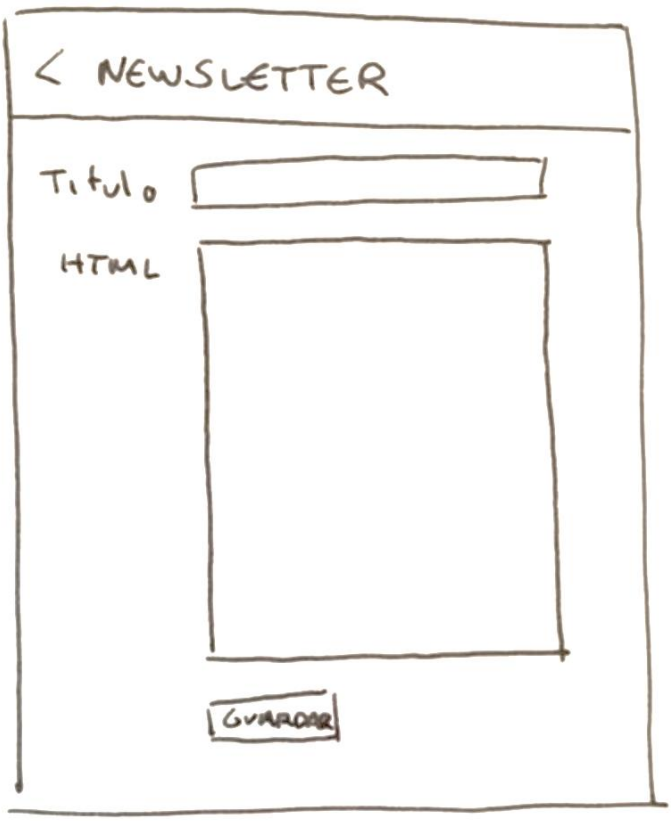
Estadísticas



Fidelización

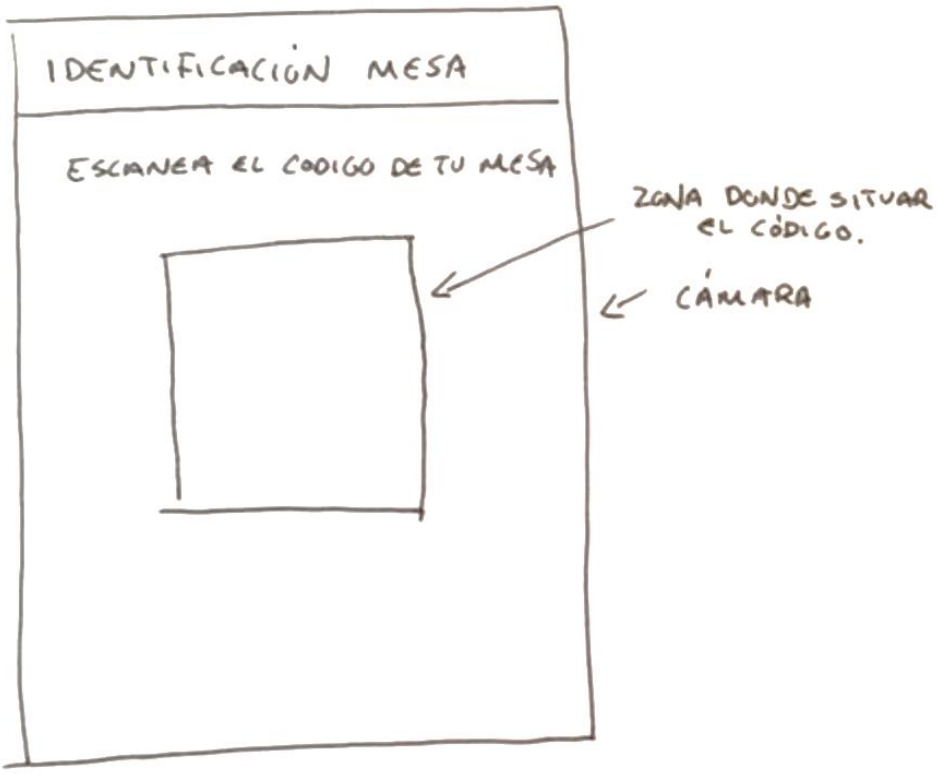


Newsletter

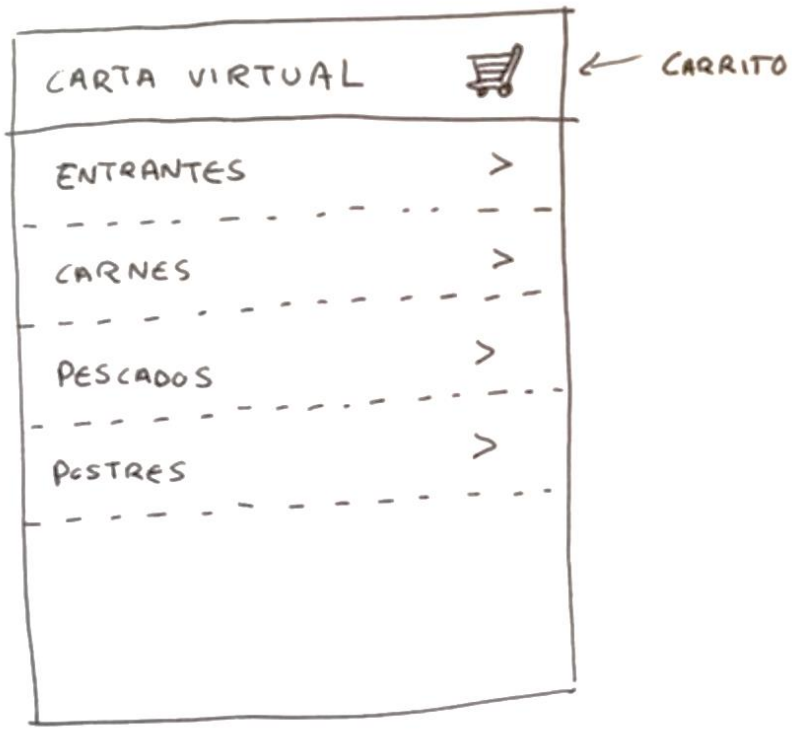


4.4.2.2 Prototipos de pantallas en el subsistema aplicación móvil

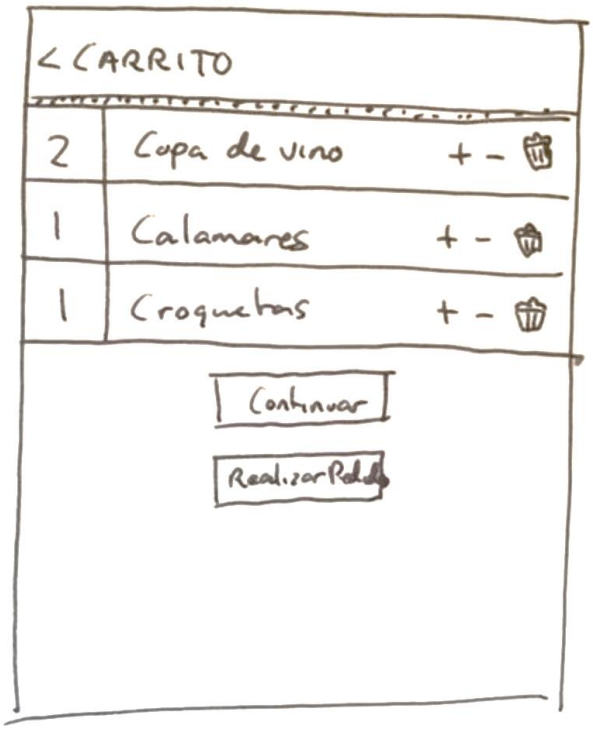
Identificación de la mesa



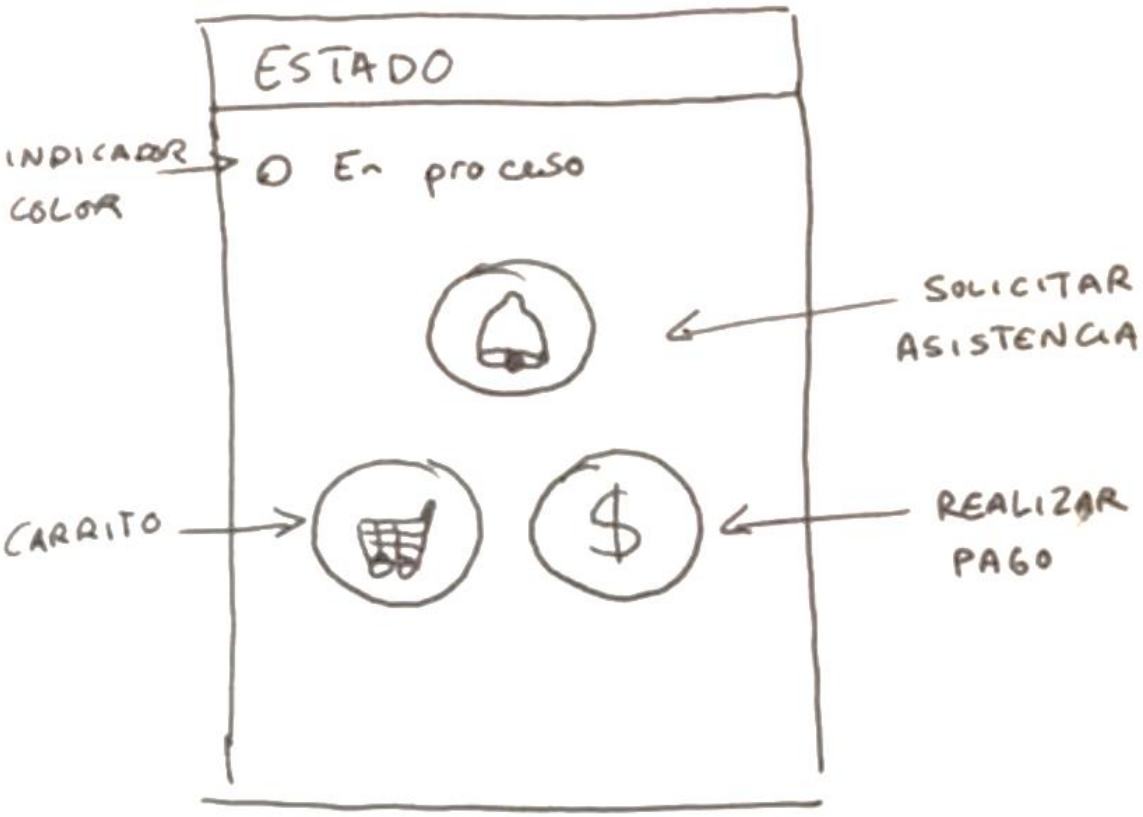
Carta Virtual



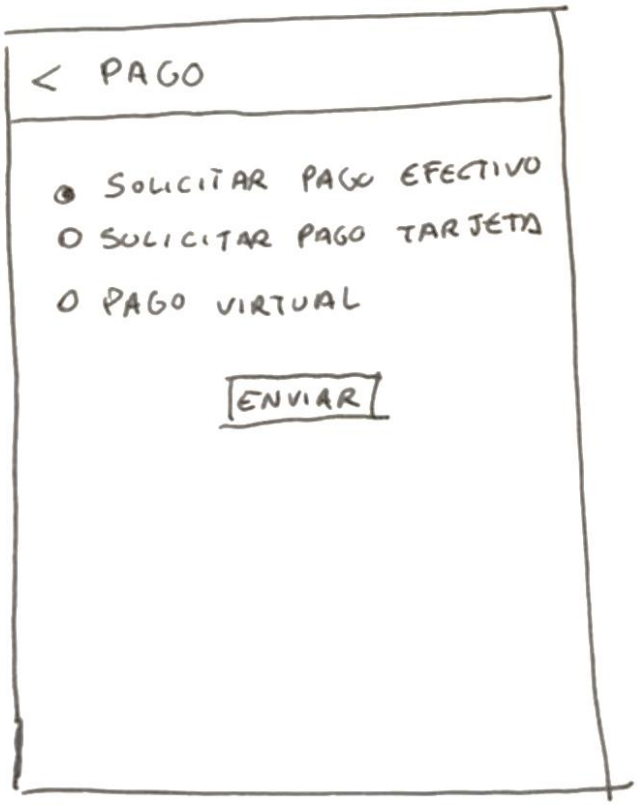
Carrito



Estado



Pago



Valoración

4.4.3 Relación de casos de uso aplicación web – Pantallas

	CU 1	CU 2	CU 3	CU 4	CU 5	CU 6	CU 7	CU 8	CU 9	CU 10	CU 11	CU 12	CU 13	CU 14	CU 15	CU 16	CU 17	CU 18
Login	X																	
Sel.Vista		X	X															
V.Barra				X	X													
V. Sala						X												
Comanda							X	X										
V.Cocina										X	X							
Cobro									X									
V. Admin												X						
Productos													X					
Producto													X					
Usuarios														X				
Usuario														X				
Estadísticas															X			
Fidelización																X	X	
Newsletter																		X

4.4.4 Relación de casos de uso aplicación móvil – Pantallas

	CU 1: Id. mesa	CU 2: Realizar pedido	CU 3: Asistencia	CU 4: Realizar pago	CU 5: Pago manual	CU 6: Pago virtual	CU 7: Valorar y comentar
Id.mesa	X						
Carta		X					
Carrito		X					
Estado			X				
Pago				X	X	X	
Valoración							X

4.5 Pruebas de aceptación

En esta sección se describen las pruebas que se han programado para validar finalmente la aplicación. Se han diseñado pruebas para cada uno de los subsistemas y se han clasificado en función de sus casos de uso para una mejor organización.

4.5.1 Pruebas de aceptación para el subsistema aplicación web para el negocio

Caso de Uso 1: Autenticarse

Caso de prueba 1	**Resultado esperado**
Loguearse en el sistema con datos válidos.	Debería aparecer la pantalla principal.

Caso de prueba 2	**Resultado esperado**
Loguearse con datos inválidos o sin rellenar ciertos campos.	El sistema debería mostrar un mensaje de error al validar el formulario, o de datos inválidos.

Caso de prueba 3	**Resultado esperado**
Loguearse con un usuario desactivado	El sistema debería mostrar un mensaje de de datos inválidos.

Caso de Uso 2: Salir

Caso de prueba 4	**Resultado esperado**
Salir manualmente	Desde la pantalla principal debería volver a la pantalla de login al pulsar sobre el botón “salir”.

Caso de prueba 5	**Resultado esperado**
Salir automáticamente.	Si eliminamos el token de sesión desde la consola javascript, debería reenviarnos a la pantalla de login.

Caso de prueba 6	**Resultado esperado**
Salir por permisos insuficientes.	Si tratamos de navegar hacia la pantalla de administración sin permisos de administrador nos debería reenviar a la pantalla de login.

Caso de Uso 3: Seleccionar vista

Caso de prueba 7	**Resultado esperado**
Seleccionar vista de cocina.	Al seleccionar esta opción se muestra la vista de cocina.

Caso de prueba 8	**Resultado esperado**
Seleccionar vista de sala.	Al seleccionar esta opción se muestra la vista de sala.

Caso de prueba 9	**Resultado esperado**
Seleccionar vista de barra.	Al seleccionar esta opción se muestra la vista de barra.

Caso de prueba 10	**Resultado esperado**
Seleccionar vista de administración.	Esta opción sólo se muestra siempre que seamos administradores. Accediendo a la vista de administración al pulsar en ella.

Caso de Uso 4: Vista de barra

Caso de prueba 11	**Resultado esperado**
Vista de barra	Se deben mostrar únicamente las bebidas a servir ordenadas por prioridad junto con las bebidas prepadas pero no servidas.

Caso de Uso 5: Marcar bebida preparada

Caso de prueba 12	**Resultado esperado**
Marcar bebida preparada.	Al pulsar sobre el botón a tal efecto, la bebida se marca de color verde (estado preparado) y se traslada del listado de bebidas a preparar a el listado de bebidas preparadas pero no servidas.

Caso de Uso 6: Vista de sala

Caso de prueba 13	**Resultado esperado**
Vista de sala	Debe mostrar el estado actual de la sala (mesas ocupadas y comensales actuales), además de una línea por cada orden con un resumen de sus elementos y la fecha de entrada.

Caso de Uso 7: Gestión de comanda

Caso de prueba 14	**Resultado esperado**
Creación de comanda	Desde la vista de sala es posible crear una comanda con al menos un producto. El estado de la orden debe ser "nuevo".

Caso de prueba 15	**Resultado esperado**
Edición de comanda	En cualquier momento desde la vista de sala es posible modificar el estado de una comanda para eliminar o añadir elementos, o cambiar parámetros.

Caso de prueba 16	**Resultado esperado**
Eliminación de comanda	Será posible eliminar una comanda desde la vista de sala como si nunca hubiese existido. Desapareciendo sus productos de cocina o barra aun estando en proceso.

Caso de Uso 8: Marca de pasos de mesa

Caso de prueba 17	**Resultado esperado**
Marcado de pasos de mesa	Es posible modificar de forma individual la prioridad de cada producto dentro de la comanda, de forma que los productos que lleguen a barra o cocina aparezcan con dicha prioridad calculada.

Caso de Uso 9: Cobro de factura

Caso de prueba 18	**Resultado esperado**
Cobro de comanda finalizada.	Únicamente cuando una comanda haya sido finalizada (todos los platos marcados como servidos) será posible realizar el cobro de la misma. Posteriormente desaparecerá de la vista de sala.

Caso de prueba 19	**Resultado esperado**
Importe de factura.	El importe de la factura debe calcularse automáticamente teniendo en cuenta el precio de los productos y las unidades de los mismos.

Caso de Uso 10: Vista de cocina

Caso de prueba 20	**Resultado esperado**
Vista de cocina	Se debe mostrar únicamente la comida a servir ordenada por prioridad junto con la comida prepada pero no servida.

Caso de Uso 11: Marcar comida como preparada

Caso de prueba 21	**Resultado esperado**
Marcar comida preparada.	Al pulsar sobre el botón a tal efecto, la comida se marca de color verde (estado preparado) y se traslada del listado de comida a preparar a el listado de comida preparada pero no servida.

Caso de Uso 12: Vista de Administración

Caso de prueba 22	**Resultado esperado**
Vista de administración	Esta vista solo debería ser visible para los usuarios administradores.

Caso de Uso 13: Gestión de productos

Caso de prueba 23	**Resultado esperado**
Creación de grupo o sección	En cualquier nodo del árbol del menú debería ser posible la creación de una nueva sección.

Caso de prueba 24	**Resultado esperado**
Creación de producto	En cualquier nodo del árbol del menú debería ser posible la creación de un nuevo producto.

Caso de prueba 25	**Resultado esperado**
Marcar producto como desactivado	Un producto podría marcarse como desactivado de forma que si apareciese en el listado desde la administración pero no lo hiciese desde el listado de productos de la comanda.

Caso de prueba 26	**Resultado esperado**
Edición de producto	Será posible modificar los detalles de un producto en cualquier momento.

Caso de prueba 27	**Resultado esperado**
Eliminación de producto o sección	Sería posible la eliminación de un producto o sección, aunque a efectos prácticos siempre sería mejor una desactivación.

Caso de Uso 14: Gestión de usuarios

Caso de prueba 28	**Resultado esperado**
Creación de usuario	Los usuarios administradores serán capaces de crear usuarios administradores o no administradores. Dichos usuarios podrán utilizarse posteriormente para loguearse en el sistema.

Caso de prueba 29	**Resultado esperado**
Edición de usuario	Será posible modificar los detalles de un usuario en cualquier momento, a excepción de su contraseña, que tendría que resetearse desde el panel de Firebase.

Caso de prueba 30	**Resultado esperado**
Desactivación de usuario	Se deberá poder desactivar un usuario invalidando así su acceso al sistema. No se podría en cambio eliminar usuarios.

Caso de Uso 15: Estadísticas

Caso de prueba 31	**Resultado esperado**
Estadísticas de ocupación	Mostrará una gráfica con el número de clientes diarias.

Caso de prueba 32	**Resultado esperado**
Estadísticas de facturación	Mostrará una etiqueda para la facturación diaria y otra para la mensual.

Caso de prueba 33	**Resultado esperado**
Estadísticas de productos populares	Mostrará una grafica con los productos más populares en orden decreciente.

Caso de prueba 34	**Resultado esperado**
Estadísticas de horarios populares	Mostrará una gráfica con el número de pedidos para cada hora del día.

Caso de Uso 16: CRM

Caso de prueba 35	**Resultado esperado**
Puntuación global	Mostrará una etiqueda con la puntuación global (estrellas).

Caso de prueba 36	**Resultado esperado**
Estadísticas de valoraciones	Mostrará una gráfica con el número de valoraciones diario.

Caso de Uso 17: Comentarios de clientes

Caso de prueba 37	**Resultado esperado**
Comentarios	Desde la misma pantalla de CRM se podrán ver los últimos comentarios de los clientes y sus valoraciones.

Caso de Uso 18: Publicar newsletter

Caso de prueba 38	**Resultado esperado**
Newsletter	Desde el panel de notificaciones de firebase podrán publicarse newsletters en forma de notificaciones, pudiendo segmentar por distintos parámetros.

Capítulo 5. Diseño del Sistema

5.1 Arquitectura del Sistema

5.1.1 Patrones de diseño

Patrones de diseño en la aplicación web para el negocio

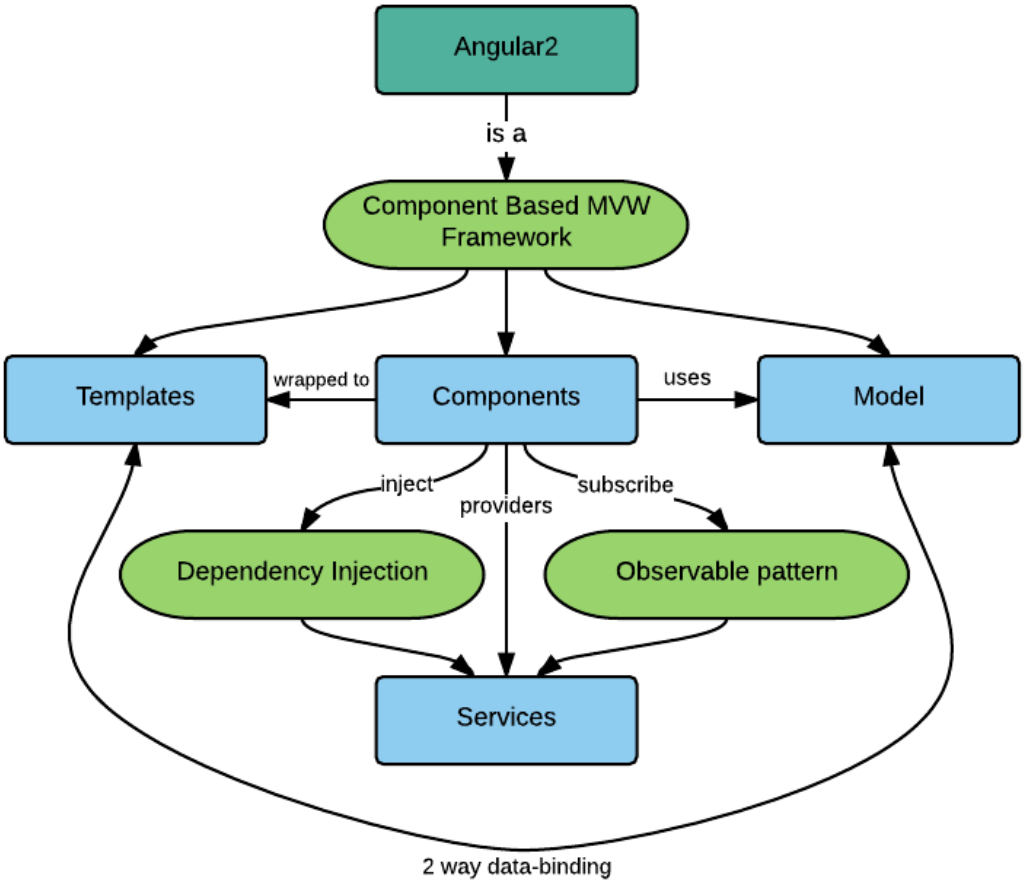


Ilustración 27 - Patrones de diseño utilizados en la aplicación web para el negocio

Patrones de diseño en la aplicación móvil para los clientes

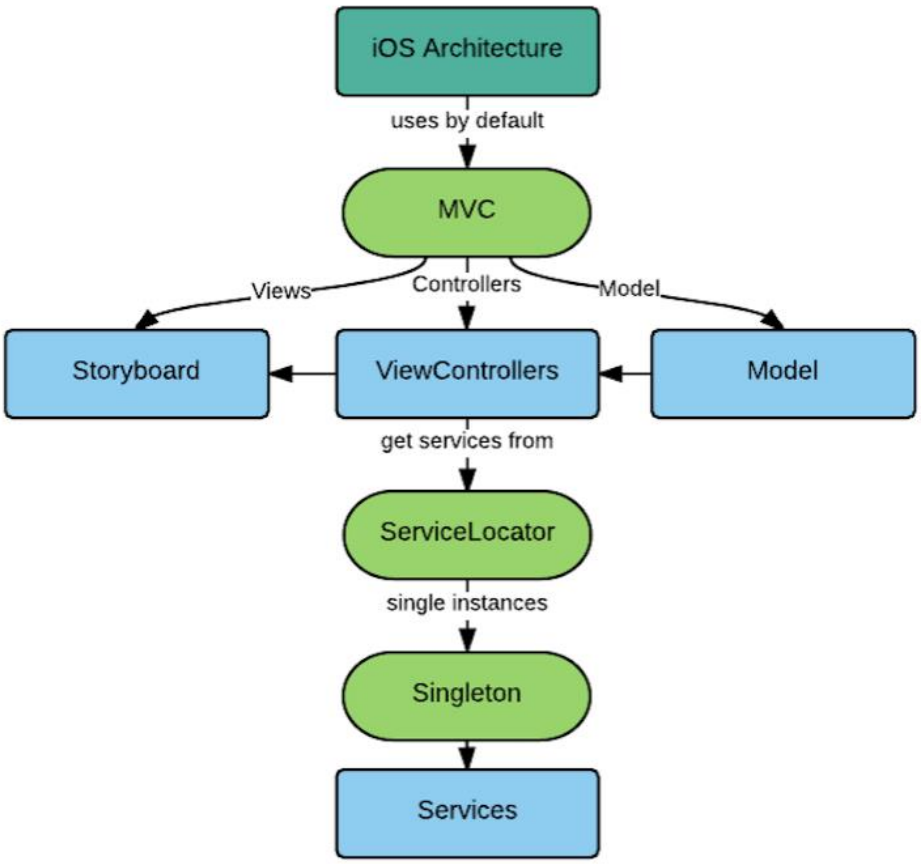


Ilustración 28 - Patrones de diseño en la aplicación móvil para los clientes

5.1.1.1 Angular 2

Angular2 es un framework de desarrollo de aplicaciones web orientado a componentes. Cada componente tiene su propia responsabilidad y está generalmente asociado a una vista dentro de la aplicación. Es por ello, que la organización del código en las aplicaciones Angular suele seguir una estructura plana asociada a las características del software antes que una división profunda típica en N-Capas.

En lugar de utilizar un patrón MVW (Model-View-Whatever) global, digamos que cada componente implementa ese patrón, al ser el componente el propio controlador para un template asociado que a su vez se encuentra ligado a un modelo.

El término MVW, también definido como MV*, hace referencia a que existe flexibilidad para los desarrolladores para elegir el formato que mejor se adopte a sus necesidades, pudiendo utilizar MVC, MVVM, MVP, o incluso combinar varios de los mismos en la misma aplicación.

Ciertas características del framework, como las extensiones reactivas, hacen que sea bastante común encontrar soluciones basadas en MVVM, antes que en MVC, como ocurre con este proyecto.

5.1.1.2 Programación reactiva y Observables

Angular utiliza algunos elementos de la programación reactiva, un paradigma que define cómo fluyen los datos internamente en la aplicación. Siguiendo este concepto, permite usar Observables para construir servicios y componentes que reaccionen a los cambios en los datos a lo largo de la aplicación.

Los observables no son una caracterísitica específica de Angular, son parte de la librería de extensiones reactivas de JavaScript (RxJS), que se incluirá como parte del propio lenguaje en la futura versión de ES7.

El propósito general de los observables es "observar" el comportamiento de una variable. De un modo imperativo, una variable cambia cuando se le añade un nuevo valor o se modifican sus datos. Este método reactivo, permite a las variables desarrollar un historial de cambios evolutivos, actualizándose de forma asíncrona con los valores de otras fuentes de datos.

La gran ventaja de la programación reactiva es la capacidad para interconectar varias fuentes de datos que automáticamente propaguen sus cambios entre sí.

5.1.1.3 Patrón MVC

El patrón MVC, (Modelo-Vista-Controlador) se utiliza en arquitectura del software para separar la lógica de negocio y los datos de una aplicación de la interfaz de usuario.

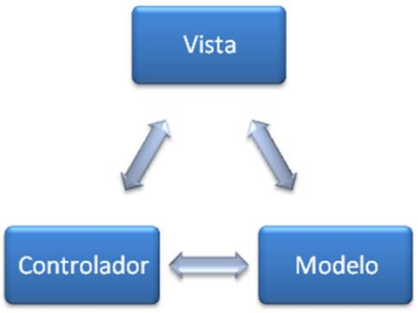


Ilustración 29 - Patrón MVC

- **Vista**: define la forma en la que se ha de presentar la información al usuario.
- **Modelo**: es la representación de los datos con los que el sistema opera.
- **Controlador**: responde a los eventos de la interfaz, haciendo de intermediario entre la vista y el modelo.

  El sistema operativo IOS utiliza este patrón por defecto, permitiendo generar las vistas y sus relaciones de una forma declarativa, con XML o visualmente sobre el Storyboard y utilizar los ViewController y NavigationController como controladores entre las vistas y el modelo.

5.1.1.4 Inyección de dependencias

La inyección de dependencias es un patrón de diseño asociado a la creación de objetos que consiste en suministrar objetos a una clase en vez de que la propia clase cree el objeto. Además, en muchas de sus facetas también se utilizan como mecanismos de inversión de control.

Angular2 fue creado usando y pensando en la inyección de dependencias y por ello incluye su propio framework para tal efecto. Todas las dependencias inyectadas con el framework de angular son singletons dentro del ámbito del inyector. Esto quiere decir, que en el caso de existir componentes anidados que utilicen las mismas dependencias que sus padres, no se crearán nuevos objetos.

5.1.1.5 Service Locator

Se trata de un patrón de diseño de software que se encarga de devolver instancias de objetos de otras clases denominadas servicios. Generalmente se implemente mediante una clase estática u a través del patrón singleton, para evitar que haya más de una instancia de la misma.

Las clases que necesiten un servicio concreto, se encargarían de llamar a un método de esta clase para obtener el mismo. Es frecuente que las clases de servicio implementen también el patrón singleton, de forma que solo sean creadas una vez.

Este patrón es utilizado en la aplicación iOS para centralizar el acceso a los servicios.

5.1.1.6 Patrón Singleton

El patrón singleton se utiliza para garantizar que una clase pueda tener una única instancia, y de ese modo, proporcional un punto de acceso global a ella.

Este patrón se implementa creando en nuestra clase un método que crea una instancia del objeto sólo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor.

En la aplicación cliente de este proyecto se utiliza este patrón para garantizar que el acceso a determinados servicios devuelva siempre la misma instancia, de modo aseguramos que nunca haya dos servicios del mismo tipo simultáneamente en ejecución.

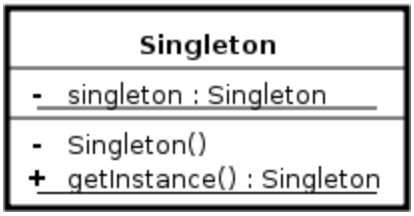


Ilustración 30 - Patrón singleton

5.1.2 Diagramas de Componentes

5.1.2.1 Diagrama de Componentes en la app web para el negocio

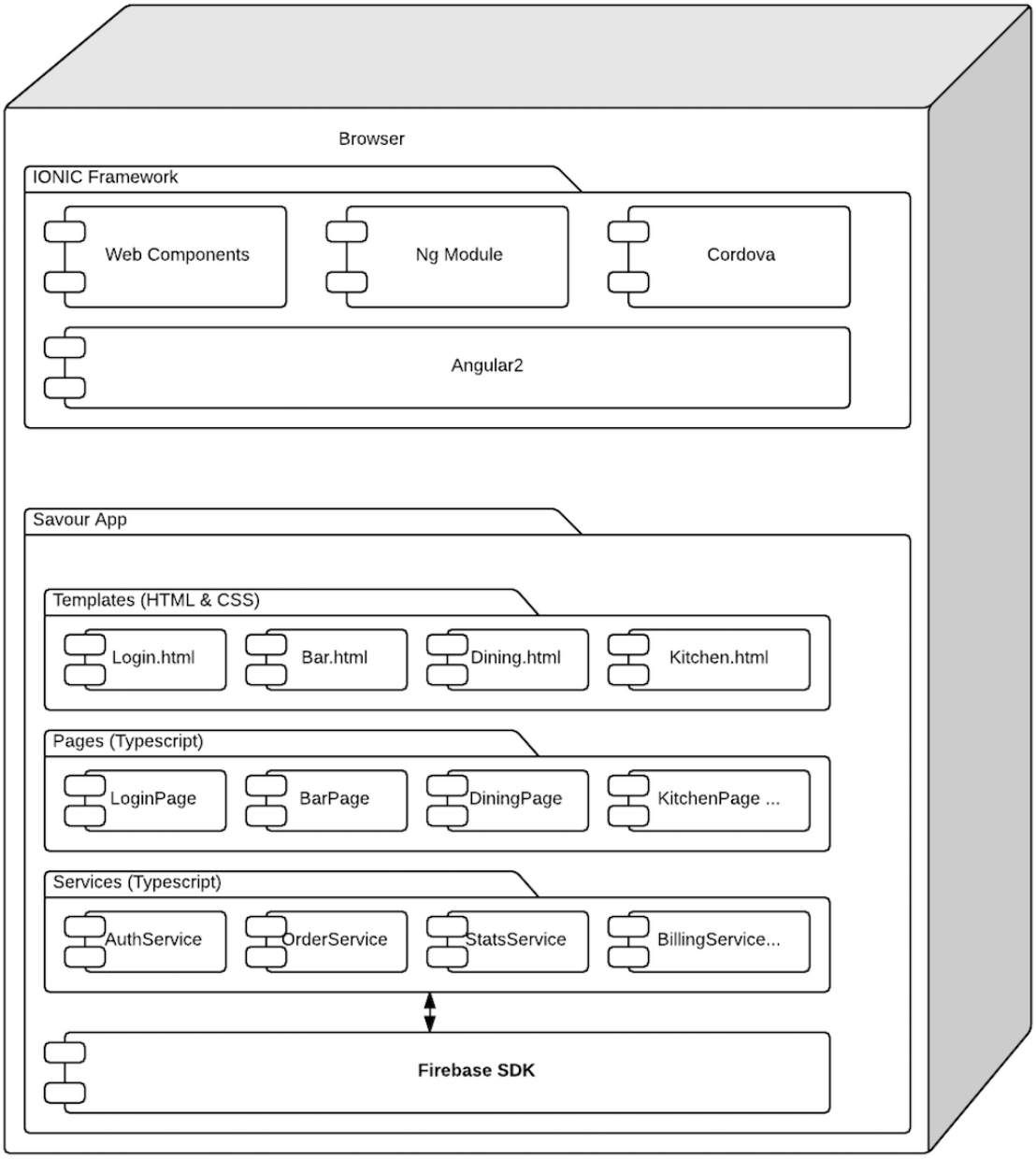


Ilustración 31 - Diagrama de componentes para la aplicación web del negocio

Ionic Framework

Proporciona una serie de componentes web responsive y con apariencia nativa para dispositivos móviles en función de su plataforma, que junto a angularJS permite la creación de aplicaciones web.

Cordova

Permite generar una aplicación móvil que utilize internamente una aplicación web a través de webviews. También proporciona módulos que se comunican con el hardware nativo del dispositivo.

NGModule

Módulo de IONIC que se encarga de asociar el framework a cualquier aplicación Angular.

Templates

Se trata de las vistas dentro de una aplicación Angular. Generalmente cada vista se encuentra asociada a un componente y es capaz de bindearse a los modelos de datos del componente de forma reactiva.

Pages

Cada una de las páginas de IONIC es un componente Angular. Actúan como controladores entre los eventos que se generan en las vistas y los datos.

Services

Clases que encapsulan la lógica de negocio de la aplicación. Generalmente son inyectadas en los componentes como métodos observables para adquirir un comportamiento reactivo de los componentes.

Firebase SDK

Framework de acceso al backend (NoSQL) y mecanismos de autenticación de Firebase.

5.1.2.2 Diagrama de Componentes en la aplicación móvil para el cliente

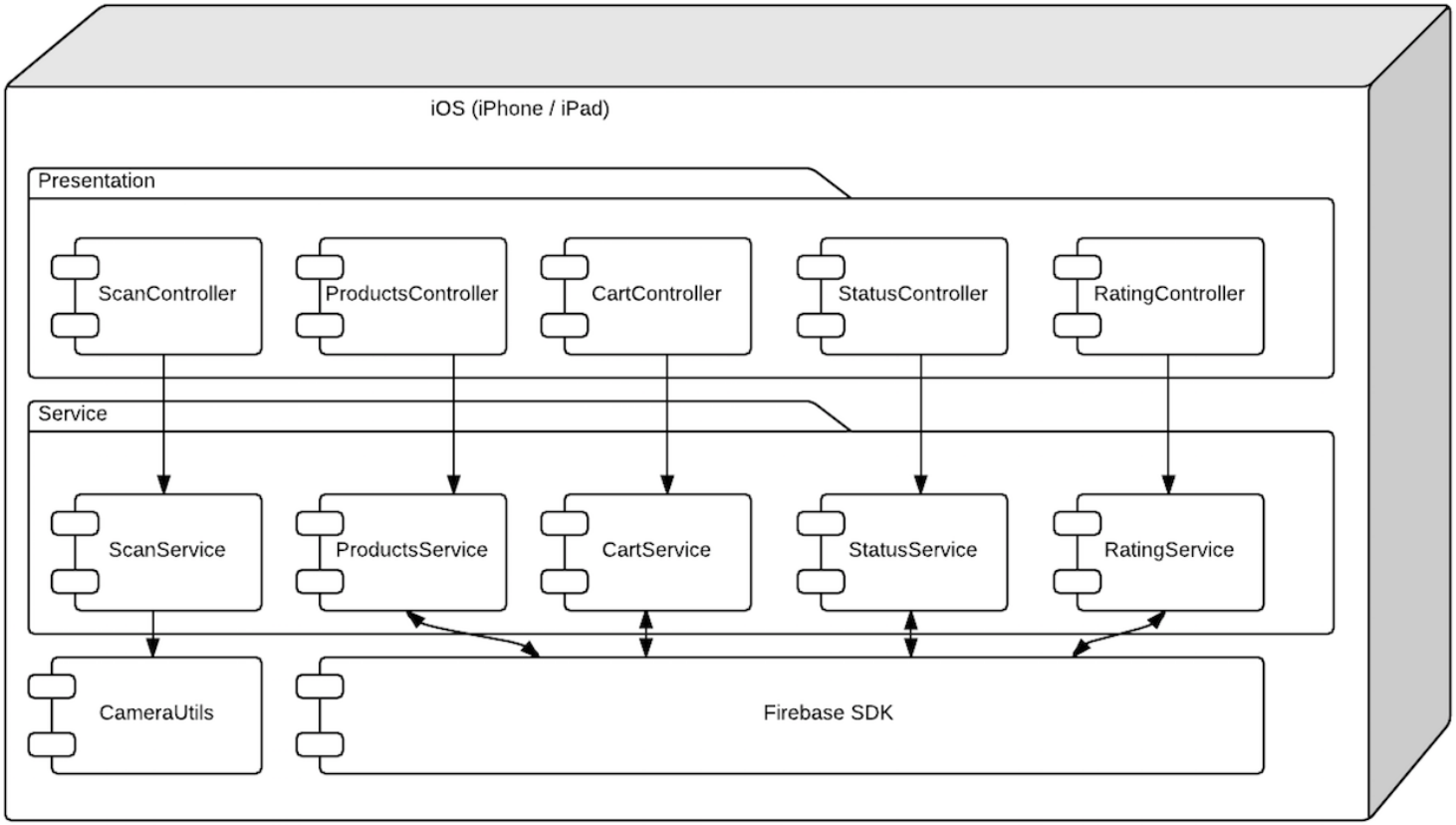


Ilustración 32 - Diagrama de componentes en la aplicación móvil para los clientes

ScanController & ScanService

Se encargan de mostrar la cámara del dispositivo e identificar la mesa respectivamente.

ProductsController & ProductsService

Una vez identificada la mesa y por consiguiente el local, deberán mostrar un listado con los productos disponibles.

CartController & CartService

Son las clases encargadas de mostrar y gestionar el carrido de la compra, así como de realizar y/o modificar los pedidos.

StatusController & StatusService

Muestran el estado de la orden actualizado en todo momento, permitiendo añadir nuevos elementos, solicitar asistencia o finalizar el pedido.

RatingController & RatingService

Permiten realizar comentarios y puntuar la experiencia una vez finalizado el pedido.

5.1.3 Diagrama de despliegue

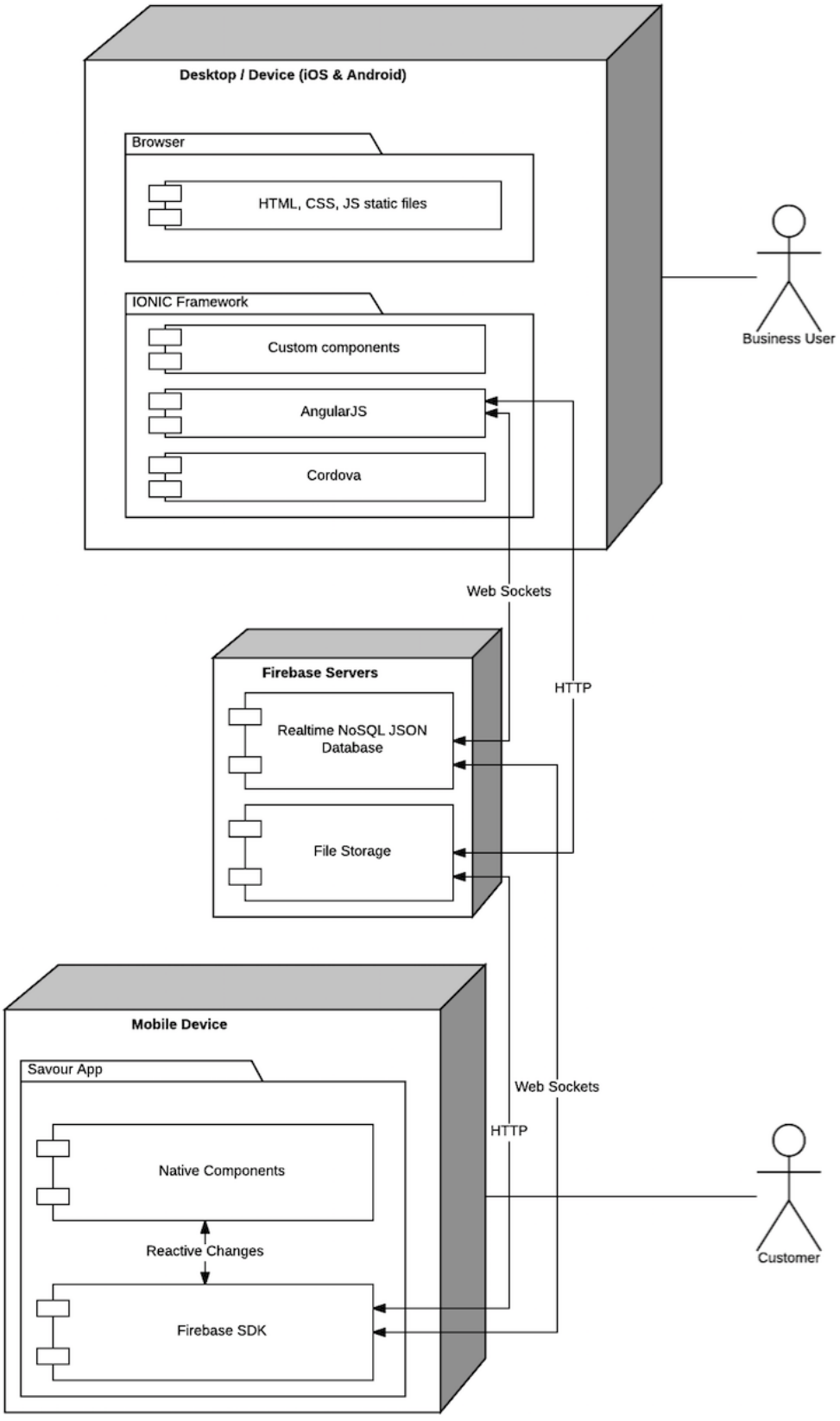


Ilustración 33 - Diagrama de despliegue de la aplicación

El diagrama anterior describe la arquitectura global del sistema. Como puede observarse, los roles de empleado y cliente acceden a dos subsistemas diferentes que internamente consumen el mismo backend, hosteado en Firebase.

Los empleados del negocio pueden utilizar una aplicación web desde cualquier navegador soportado, o utilizar las aplicaciones móviles que, internamente y gracias a la ayuda de Apache Cordova, cargan las mismas aplicaciones web en webviews nativos.

Por otro lado, los clientes utilizarían una aplicación móvil completamente nativa, implementando el sdk de firebase para su plataforma concreta, que sólo expondría cierta funcionalidad del backend como el listado de la carta, o la creación de comandas, junto con otros servicios complementarios.

Destacar que el backend se constituye por 2 módulos bien diferencidos, una base de datos NoSQL (Firebase Database), y un servidor de archivos en la nube (Firebase Storage). El primero se utiliza para guardar los propios datos de la aplicación, mientras que el segundo es un complemento para almacenar datos de formato binario, que se linkearían desde Firebase Database para obtener mayor rendimiento.

Aclarar que, a diferencia de muchos backends en la nube, la característica que hace muy interesante a Firebase es utilizar Websockets como canal de comunicación, por lo que los datos viajan mucho más rápidos al no tener que volver a efectuar nuevas conexiones, evitando las latencias, redundancia de cabeceras, etc.

Sin embargo, Firebase Storage utiliza el protocolo HTTP para la transferencia de datos binarios, pues al ser previsiblemente ficheros de mayor tamaño no se establecerán tantas conexiones.

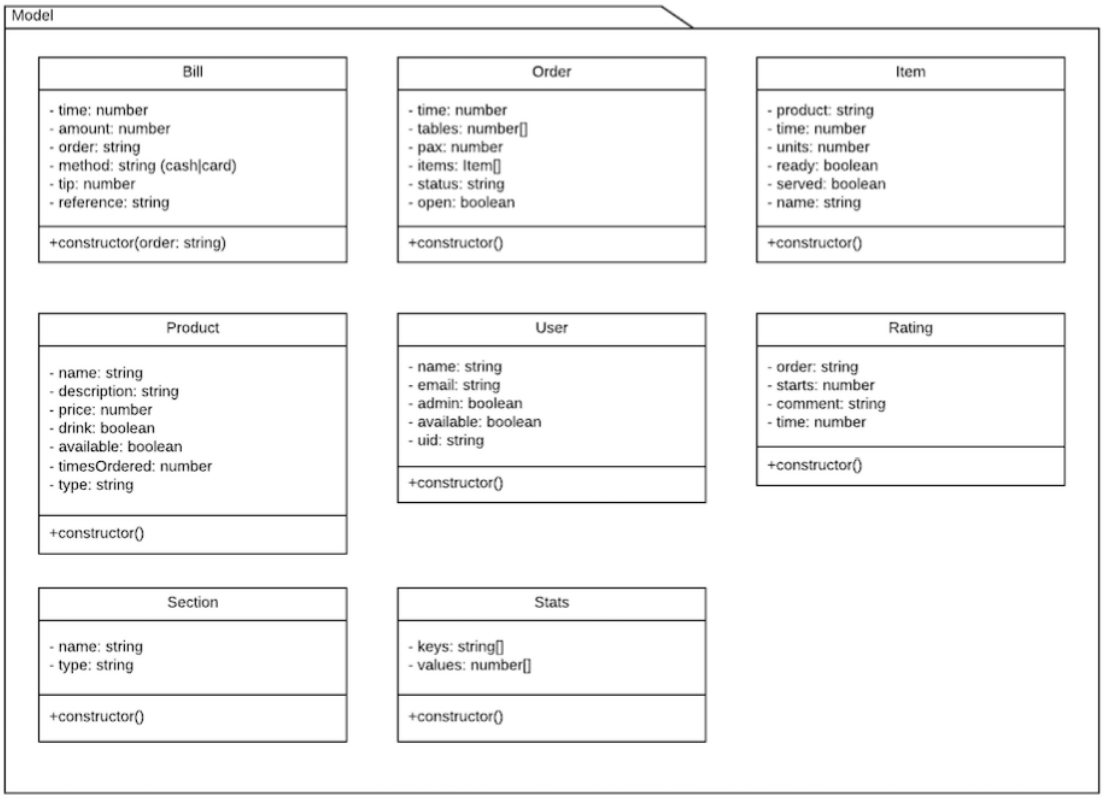
5.2 Diseño de Clases

5.2.1 Diagramas de paquetes y clases importantes

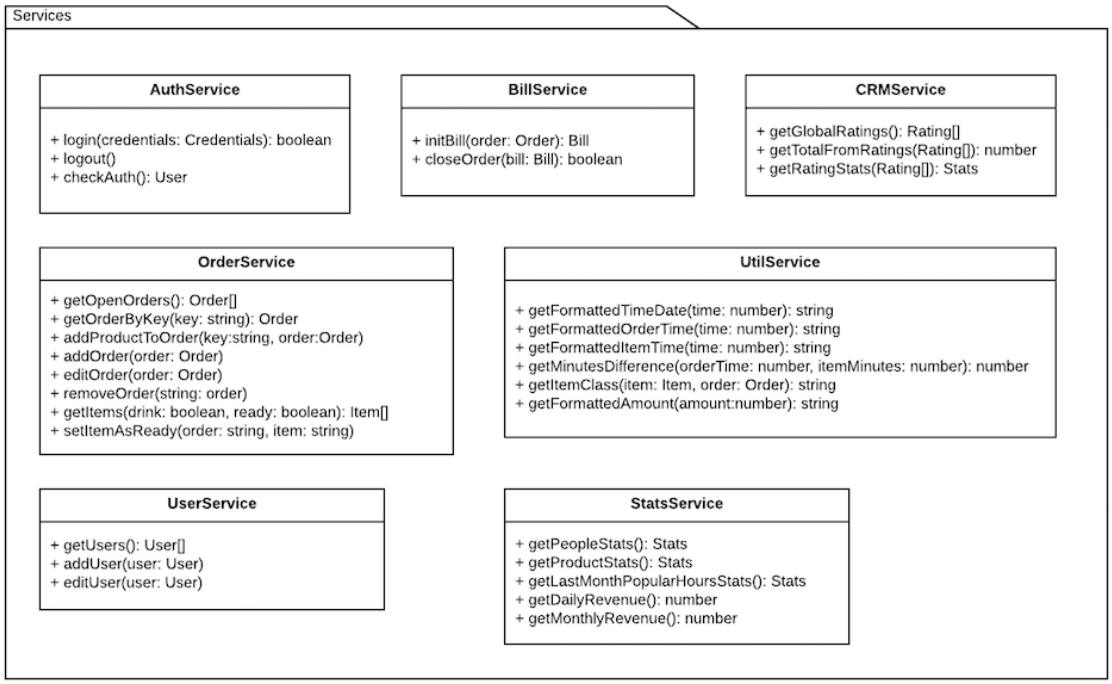
A continuación se presentan los diagramas de las clases más importantes de ambas aplicaciones de las que fue necesario realizar un diseño previo a la implementación.

5.2.1.1 Diagramas de clases en el subsistema aplicación web para el negocio.

Paquete Modelo

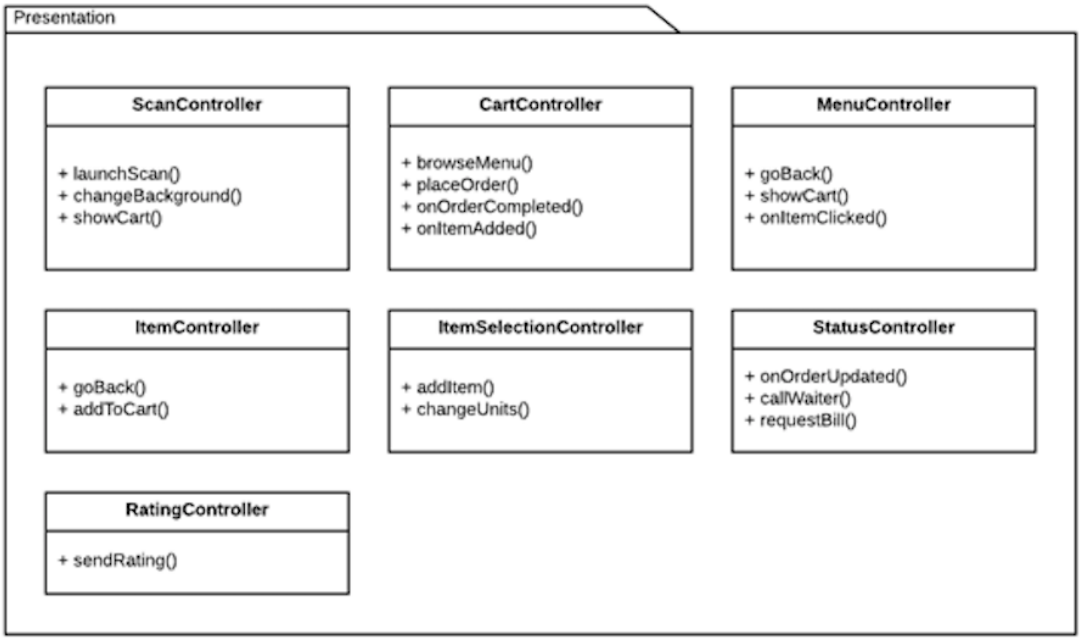


Paquete Servicios

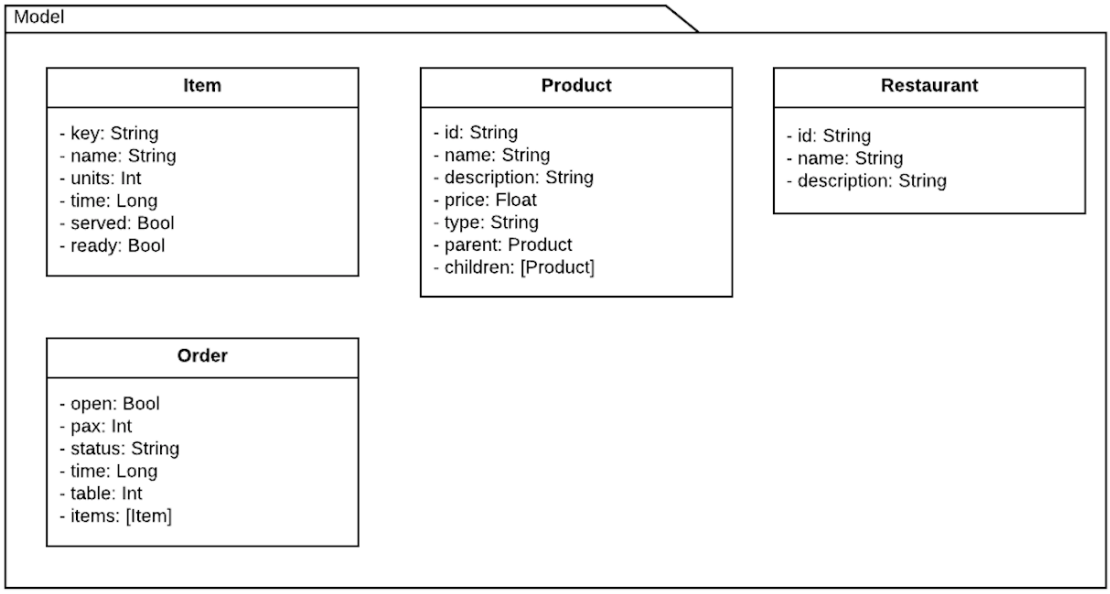


5.2.1.2 Diagramas de clases en el subsistema aplicación móvil para el cliente

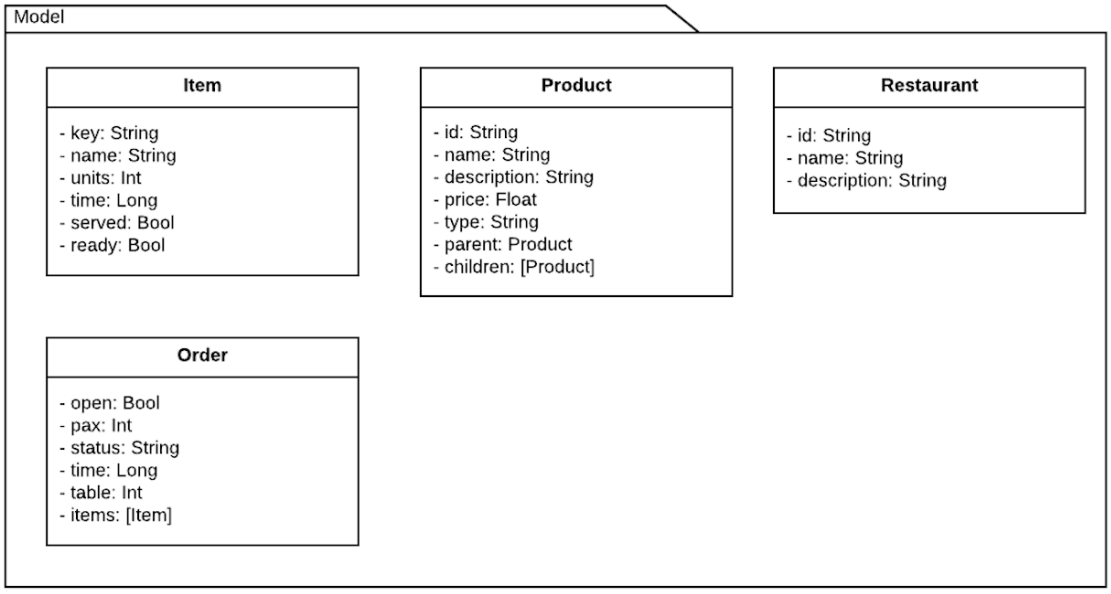
Paquete presentación



Paquete modelo



Paquete servicios



5.3 Diagramas de secuencia.

5.3.1 Autentificación en el sistema

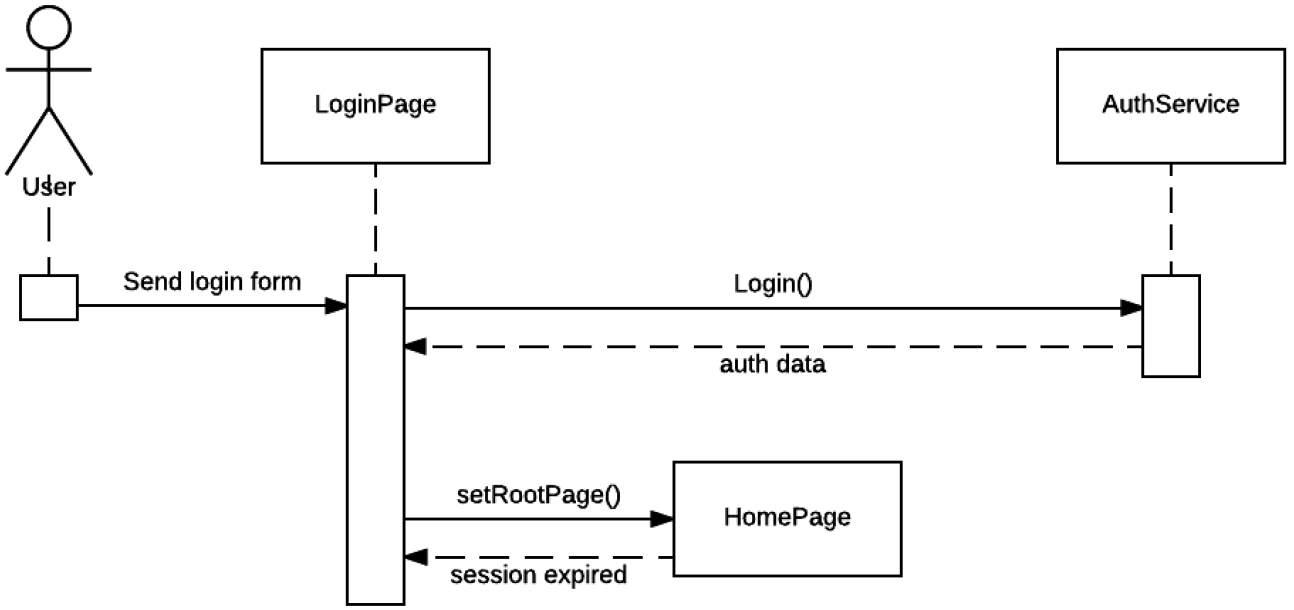


Ilustración 34 - Diagrama de secuenta para la autentificación en el sistema

5.3.2 Creación de comanda

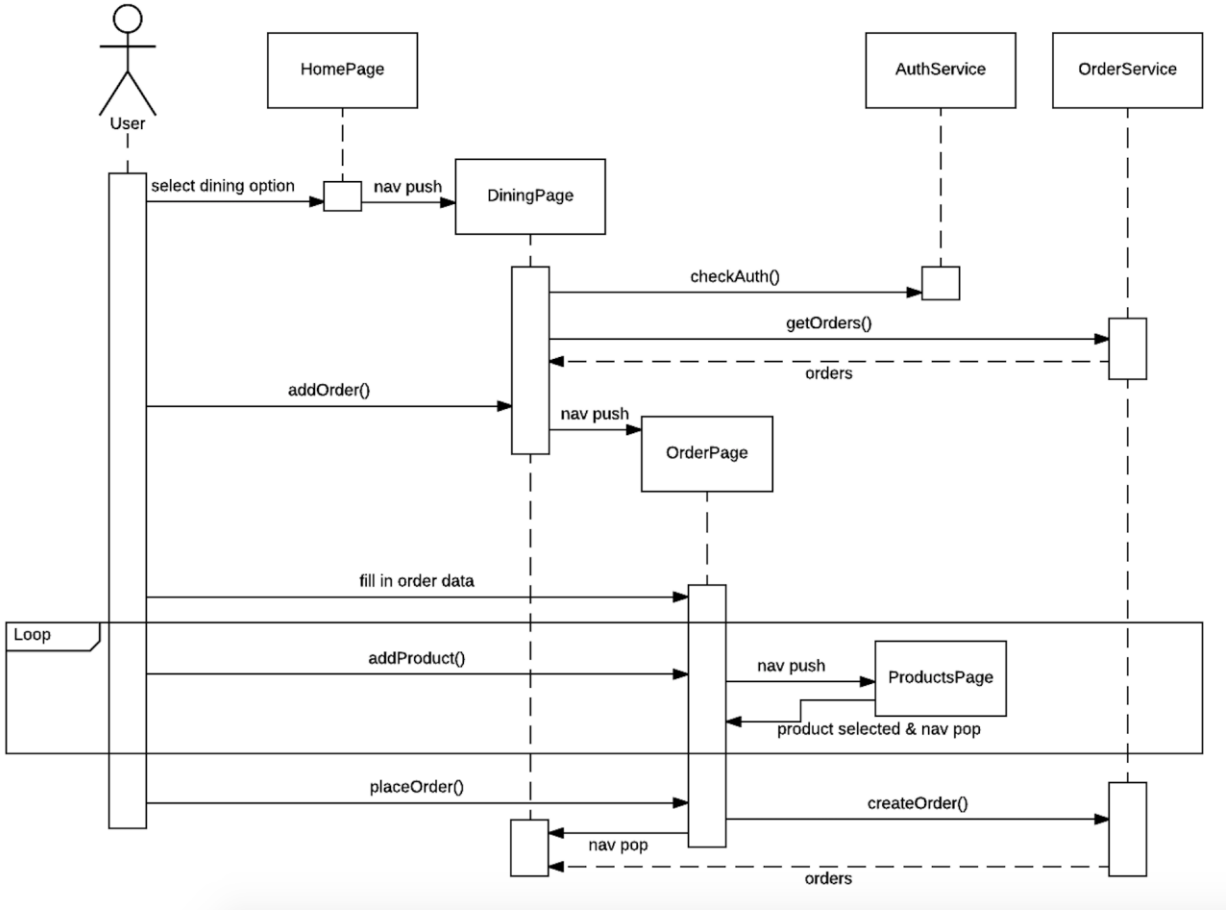


Ilustración 35 - Diagrama de secuencia para la creación de una comanda

5.3.3 Marcar bebida como preparada

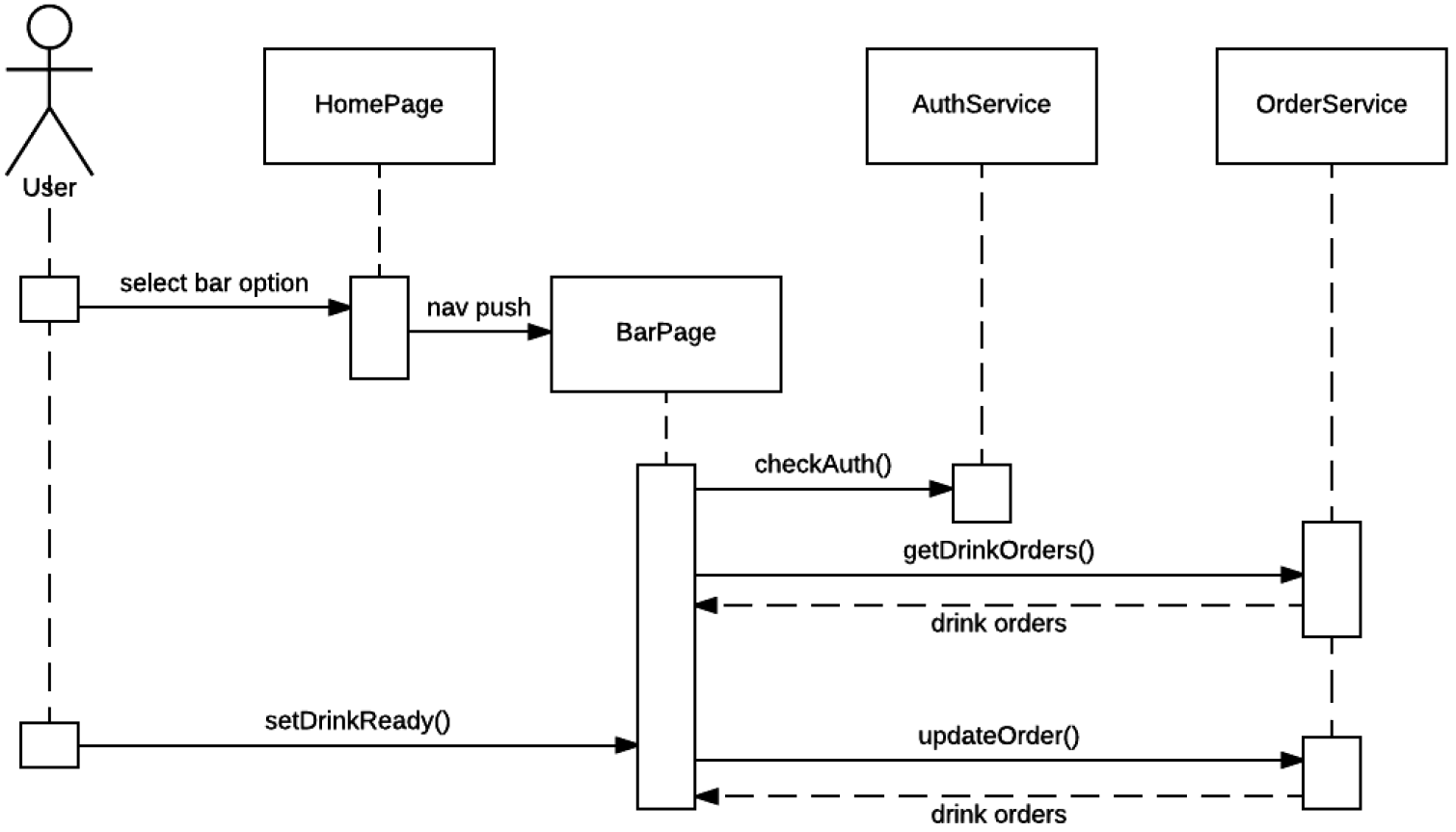


Ilustración 36 - Diagrama de secuencia para marcar bebida como preparada

5.3.4 Marcar comida como preparada

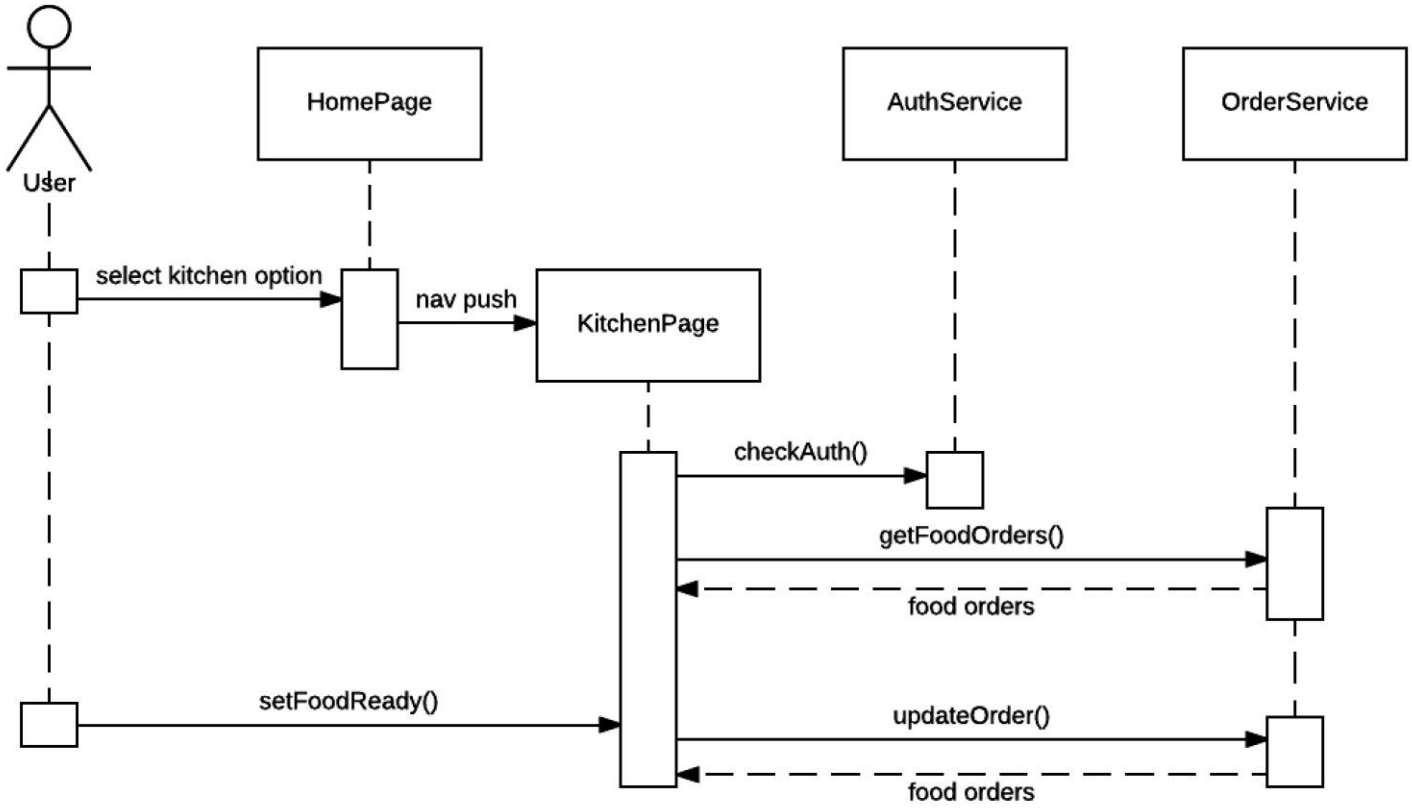


Ilustración 37 - Diagrama de secuencia para marcar comida como preparada

5.3.5 Cobro de comanda

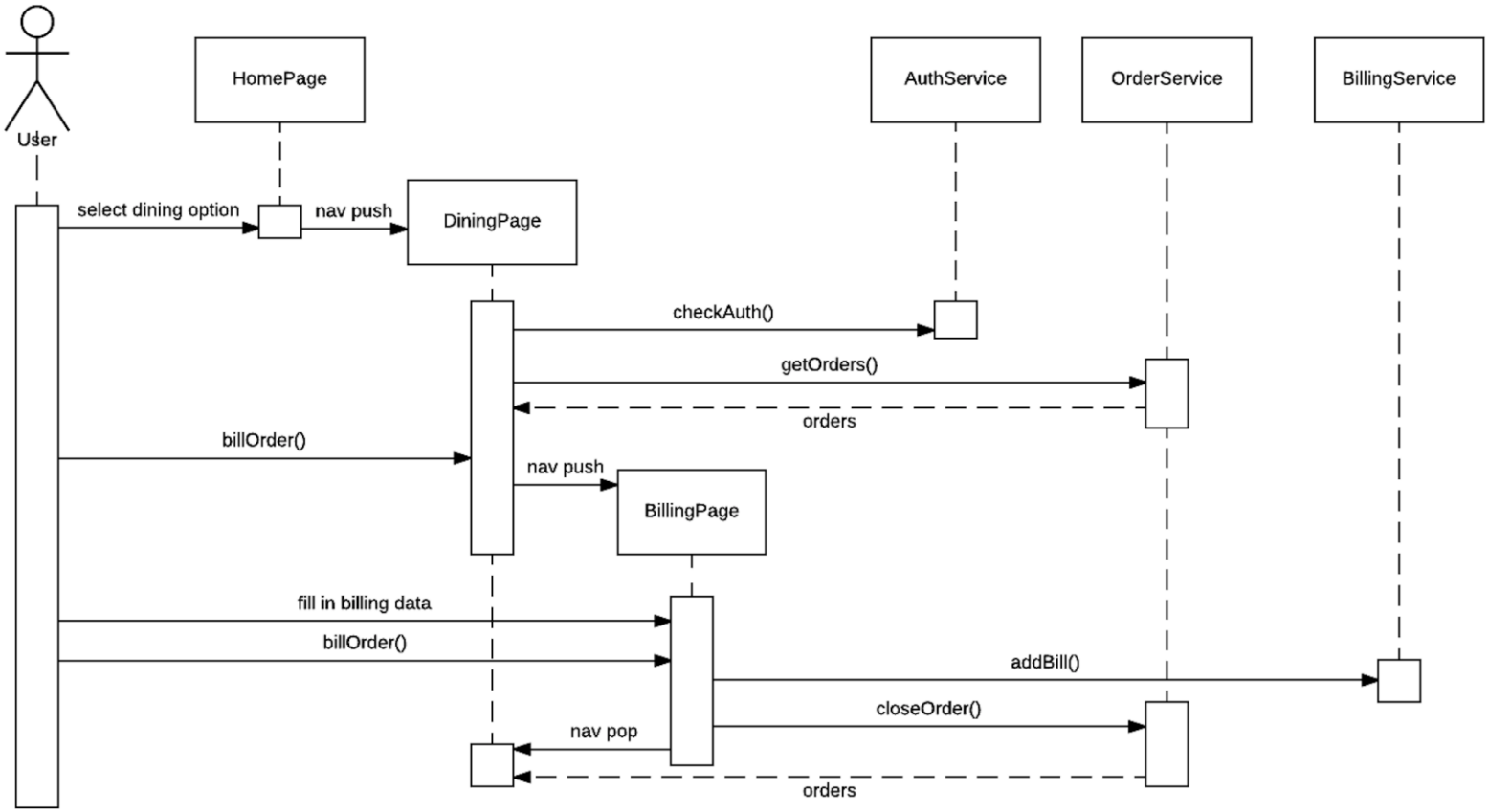


Ilustración 38 - Diagrama secuencia para el cobro de una comanda

5.4 Diseño de la Base de Datos

5.4.1 Base de datos realtime de Firebase

Para el desarrollo de este proyecto se ha utilizado una base de datos JSON (NoSQL) hosteada en la nube por Firebase. La gran ventaja de esta base de datos, además de la escalabilidad automática, reside en los mecanismos que firebase proporciona para el acceso y la sincronización de sus datos.

En lugar de utilizar las típicas peticiones HTTP a la nube, Firebase utiliza **websockets**, estableciendo una conexión activa para cada cliente conectado, y sincronizando los cambios de información en tiempo real. Esta funcionalidad la hace realmente útil en combinación con las características reactivas de Angular2 y encaja perfectamente en este proyecto, pues uno de los objetivos buscados era que el personal del restaurante no tuviese que refrescar el navegador para obtener los cambios en los datos, y otras alternativas como AJAX podrían resultar demasiado pesadas.

Además, otra de sus características interesantes es que **puede funcionar offline** durante pequeños periodos de tiempo, cacheando la información en memoria o en disco, y publicándola una vez que se reestablezca la conexión.

Pese a sus similitudes con otras bases de datos NoSQL como MongoDB, Firebase cuenta con algunas limitaciones como el no soportar Arrays, aunque internamente los emula como objetos, siendo los índices del array las claves del objeto.

5.4.2 Estructura de la base de datos

A continuación, se muestra la estructura general del árbol JSON que constituye la base de datos. Se han utilizando llaves "{}" para representar los identificadores de los objetos, y los tipos de datos se muestran como los valores de las claves para cada una de ellas.

```
{
    "businesses": {
        "{business_id}": {
            "name": "string",
            "description": "string"
        }
    },
    "users": {
        "{business_id}": {
            "{user_id}": {
                "uid": "string",
                "name": "string",
                "email": "string",
                "available": "boolean",
                "admin": "boolean"
            }
        }
    },
    "sections": {
        "{business_id}": {
            "{section_id}": {
                "name": "string",
                "sections": "[section]",
                "products": "[string (product_id)]"
            }
        }
    },
    "products": {
        "{business_id}": {
            "{product_id}": {
                "name": "string",
                "description": "string",
                "drink": "boolean",
                "available": "boolean",
                "price": "number",
                "timesOrdered": "number"
            }
        }
    },
    "orders": {
        "{business_id}": {
            "{order_id}": {
                "pax": "number",
                "tables": "[number]",
                "time": "number (epoch)",
                "open": "boolean",
                "status": "string",
                "items": [{
                  "product": "string (product_id)",
                  "meta": {
                      "ready": "boolean",
                      "served": "boolean",
                      "units": "number",
                      "time": "number (epoch)"
                  }
                }]
            }
        }
    },
```

```
    "bills": {
        "{business_id}": {
            "{bill_id}": {
                "order": "string (order_id)",
                "time": "number (epoch)",
                "method": "string",
                "reference": "string",
                "amount": "number",
                "tip": "number"
            }
        }
    },
    "ratings": {
        "{business_id}": {
            "{rating_id}": {
                "order": "string (order_id)",
                "stars": "number",
                "comment": "string",
                "time": "number (epoch)"
            }
        }
    }
}
```

Como puede observarse, la base de datos sigue una estructura FLAT, tal y como recomiendan desde las buenas prácticas de Firebase. Los motivos principales para llevar a cabo una estructura de este estilo son:

- **Límite de profuncidad**: Actualmente firebase permite un máximo de 32 niveles de profuncidad. Puede parecer más que suficiente, pero en el caso de utilizar estructuras demasiado anidadas sin algún tipo de control, es posible llegar fácilmente hasta dicho límite provocando errores inesperados.

- **Lectura obligada de todo el nodo**: Otra de las limitaciones de utilizar una estructura anidada se encuentra en que, para obtener un simple valor de un nodo superior, firebase obliga a descargar todos los nodos interiores, lo cual es una sobrecarga innecesaria.

También existen ciertas limitaciones en consultas. No es posible realizar filtrado de los datos (como si de una claúsula WHERE se tratase) a más de un valor, y los tipos de filtrado soportados únicamente son:

- **EqualTo**: por valor.

- **StartAt**: empezando por.

- **EndAt**: acabando por.

Junto con la posibilidad de ordenar los datos, y limitarlos en número.

Por este hecho, es de especial importancia tratar de evitar las consultas muy complejas. Una posible solución sería almacenar datos precalculados durante las inserciones y actualizaciones. Un ejemplo sería el campo "**timesOrdered**" de los productos. Este campo contiene un contador con el número de veces que ha realizado el pedido de un producto, y que se utiliza para las estadísticas.

Cada vez que se realiza un nuevo pedido de un producto, una transación se encarga de incrementar su valor. De no existir este campo precalculado, sería necesario descargarse todo el árbol de pedidos (orders) e iterar por cada uno de ellos contando el número de productos iguales. Algo que para este tipo de bases de datos sería inviable.

5.4.3 Autentificación, índices y seguridad en los datos

Autentificación

Firebase proporciona mecanismos de autentificación para el acceso a su base de datos. Conviene recordar que, al tratarse de una base de datos en la nube, cualquier persona con el enlace al diccionario de datos podría descargar y/o modificar la información sin consentimiento.

Entre los mecanismos de autentificación que proporciona firebase se encuentran:

- **Anónima**: A través de un token generado para cada sesión.
- **Email y contraseña**: Como mecanismo clásico, permitiendo dar de alta nuevo usuarios desde la consola de firebase o a través de su API.
- **Redes sociales**: Facebook, Twitter, Github, Google.

Para esta aplicación se ha utilizado Email y Contraseña, pues es el método que mejor encaja para un negocio. Se proporcionaría un usuario administrador que podría crear nuevos usuarios desde la propia aplicación, y que internamente se dan de alta en Firebase a través de su API.

El campo "uid" del árbol de usuarios, hace referencia al identificador del usuario para la autentificación en firebase. Cuando un usuario inicia sesión, la aplicación recibe dicho UID, y posteriormente trata de identificar al usuario realizando una consulta a la base de datos.

Índices

Resulta curioso hablar de índices en bases de datos NoSQL, más aún cuando existe tan poca flexibilidad a la hora de filtrar los datos. Pues bien, Firebase no ofrece sólo la posibilidad de añadir índices a los datos, sino que también incorpora 2 índices por defecto: clave y prioridad.

Ambos índices tienen que ver con los métodos de generación automática de claves de Firebase. Cuando hacemos push a un objeto JSON para un nodo concreto, si no especificamos una clave, Firebase se encarga de generar una cadena aleatora del siguiente tipo:

```
-KJWcNULOuCLB80LTWA_
```

A simple vista puede parecer un string aleatorio, pero en realidad guarda mucho sentido pues se trata de una clave **única** y con un **timestamp** interno.

Esto permite indexar unívocamente los nodos, a la vez que conocer su fecha de inserción que se utiliza internamente para la resolución de conflictos en los datos, permitiendo además realizar ordenaciones por fecha (prioridad).

Para el desarrollo de esta aplicación se utilizan varios índices como uid de usuarios, fecha de factura, fecha y estado de pedido, etc.

Seguridad en los datos

La autentificación de firebase no serviría de nada si no viene acompañada de un mecanismo de autorización. Por suerte, es posible establecer una serie de reglas en la base de datos que no sólo permiten definir el control de acceso para cada uno de los nodos o claves del árbol, sino que también proporciona la posibilidad de incluir restricciones de integridad y validación de los datos.

A continuación, se muestra un fragmento de estas reglas para el nodo de valoraciones de los usuarios, que se explican posteriormente.

```
"ratings": {
    ".read": "auth !== null",
    ".write":"auth !== null",
    ".validate": "newData.hasChildren(['order', 'stars', 'time'])",
    "order": {
      ".validate": "newData.isString() && root.child('orders/' + newData.val()).exists()"
    },
    "stars": {
      ".validate": "newData.isNumber() && newData.val() >= 0 && newData.val() <= 5"
    },
    "comment": {
      ".validate": "newData.isString() && newData.val().length < 300"
    },
    "time": {
      ".validate": "newData.isNumber()"
    }
}
```

Las primeras reglas que aparecen (.read, .write) corresponden a la autorización de lectura y escritura del nodo de valoraciones. Como puede observarse, se está restringiendo el acceso únicamente a usuarios autenticados en el sistema. Se podría ir mucho más alla, por ejemplo, permitiendo el acceso exclusivo a los administradores o a los usuarios que haya sido responsables de la creación de la órden asociada, pero no se ha considerado necesario.

El resto de reglas corresponden a las restricciones de integridad, que pueden resumirse en las siguientes:

- Una valoración debe tener al menos un pedido asociado, la puntuación en estrellas y la fecha (epoch) de creación, siendo los comentarios opcionales.
- El identificador del pedido debe ser una cadena y existir en el nodo correspondiente de pedidos.
- La puntuación en estrellas debe ser un número entre 0 y 5 (ambos inclusive).
- En el caso de incluir un comentario, debe ser una cadena inferior a 300 caracteres.
- La fecha debe ser un número, representando los milisegundos pasados desde el 01/01/1970.

Cada uno de los nodos tienen su propio conjunto de reglas personalizadas. Cabe destacar que establecer un control de acceso mínimo es esencial, pues por defecto Firebase permite el acceso público de lectura y escritura a la raíz de la base de datos.

5.5 Diseño de la Interfaz

En esta sección se muestra una comparativa entre el diseño de la interfaz final que se mostró en la fase de análisis y la interfaz final de la aplicación. Finalmente se muestra una tabla de trazabilidad entre los casos de uso y las pantallas finales de la aplicación.

5.5.1 Diseño de la interfaz en el subsistema aplicación web para el negocio

Login

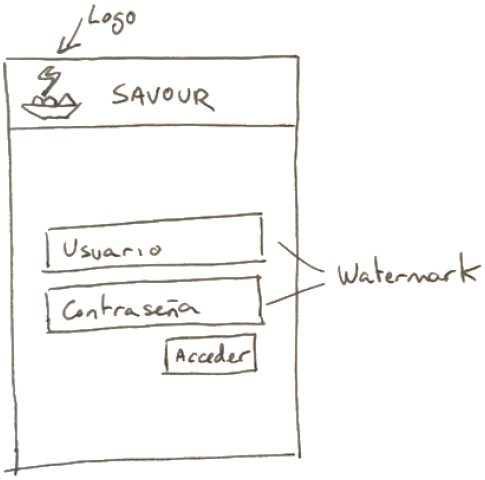


Ilustración 40 - Prototipo pantalla login

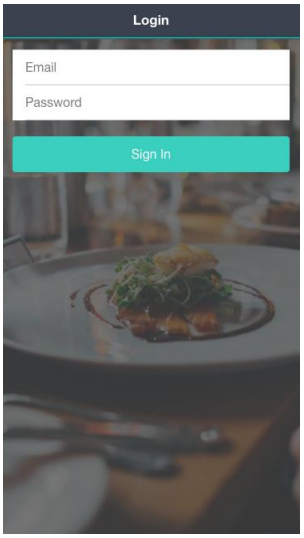


Ilustración 39 - Pantalla final login

Selección de vista

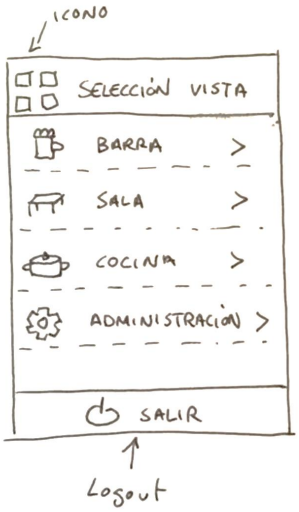


Ilustración 42 - Prototipo selección vista

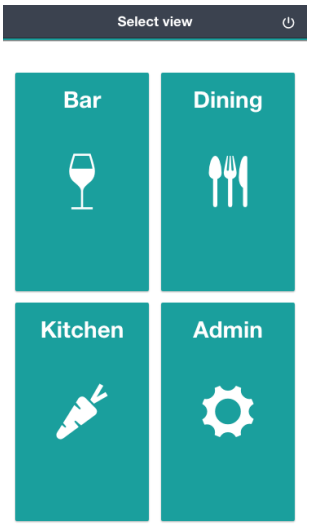


Ilustración 41 - Pantalla final selección vista

Vista de barra

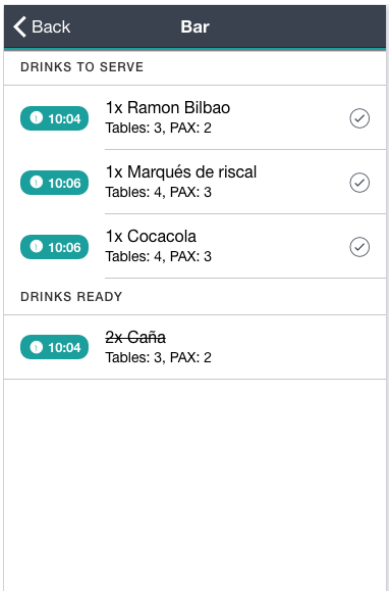


Ilustración 43 - Prototipo Pantalla Vista de barra

Ilustración 44 - Pantalla final vista de barra

Vista de sala

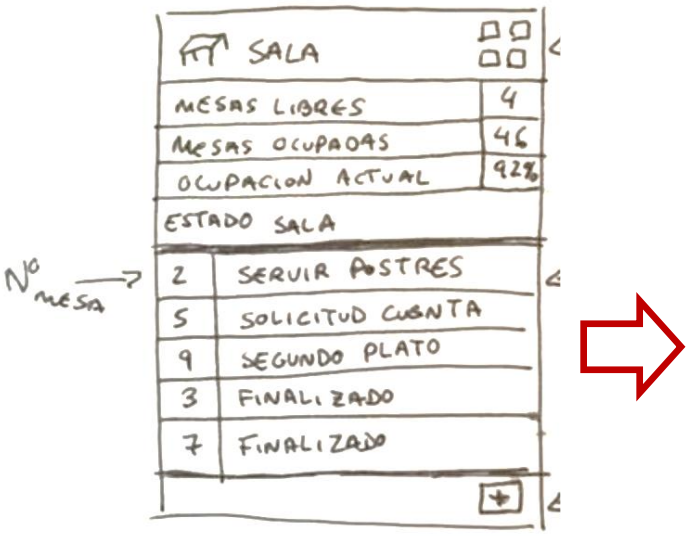


Ilustración 45 - Prototipo vista de sala

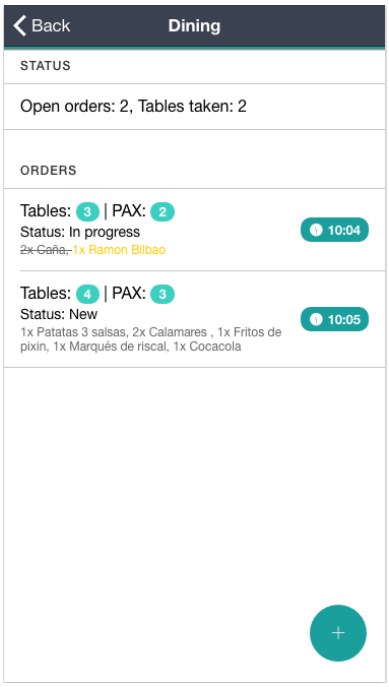


Ilustración 46 - Pantalla final vista de sala

Comanda

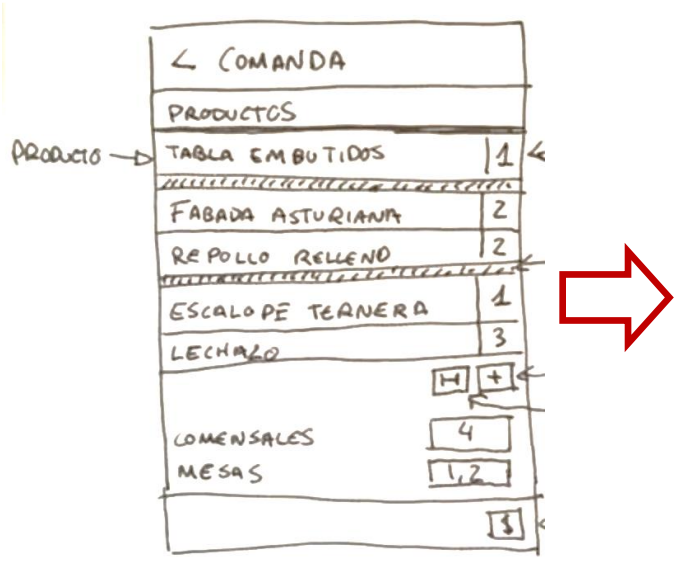


Ilustración 47 - Prototipo pantalla de comanda

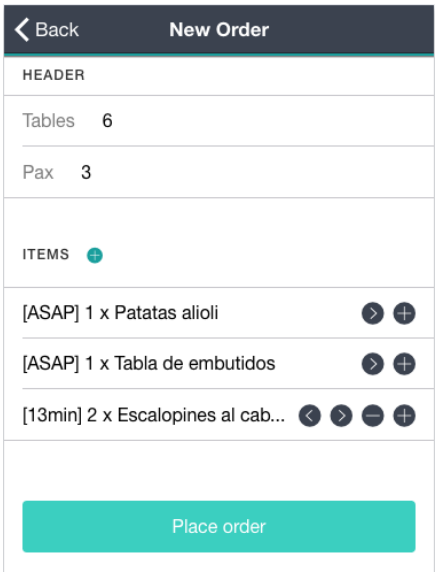


Ilustración 48 - Pantalla final comanda

Vista de cocina

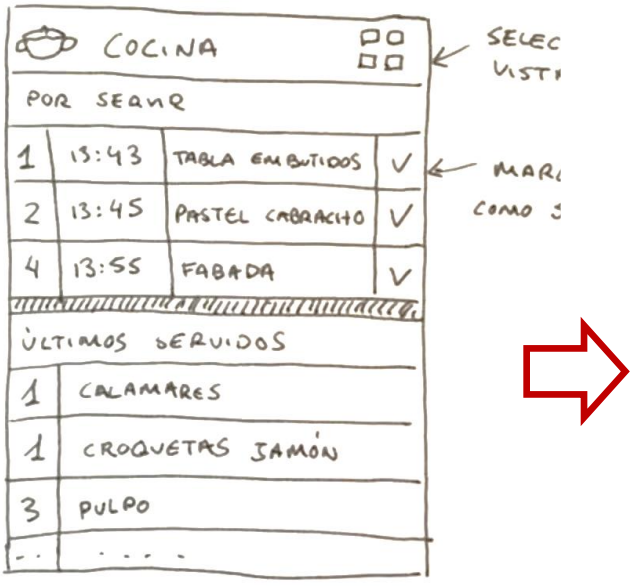


Ilustración 49 - Prototipo vista de cocina

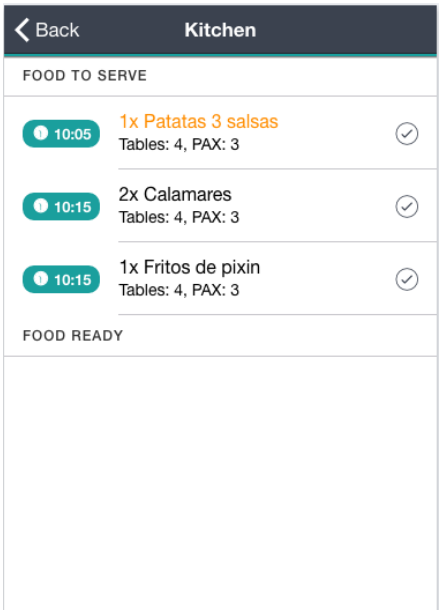


Ilustración 50 - Pantalla final vista de cocina

Cobro

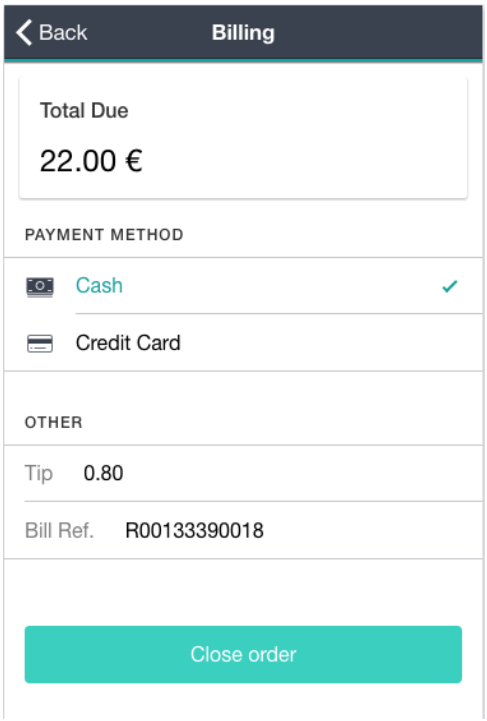
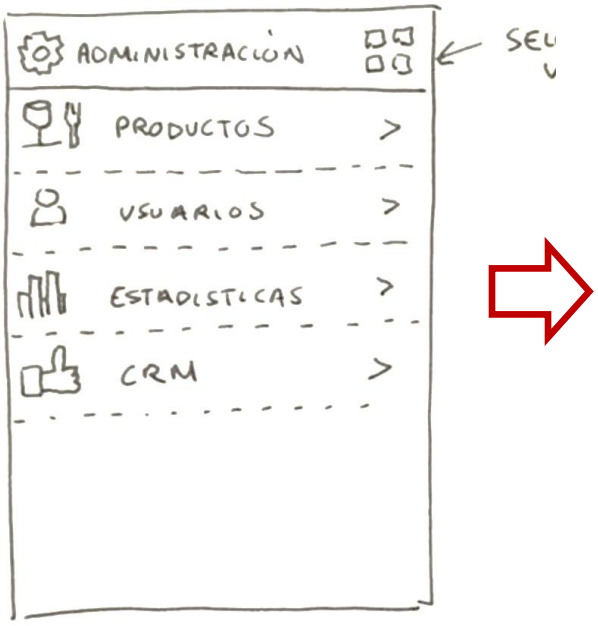


Ilustración 51 - Prototipo pantalla cobro

Ilustración 52 - Pantalla final cobro

Vista de administración



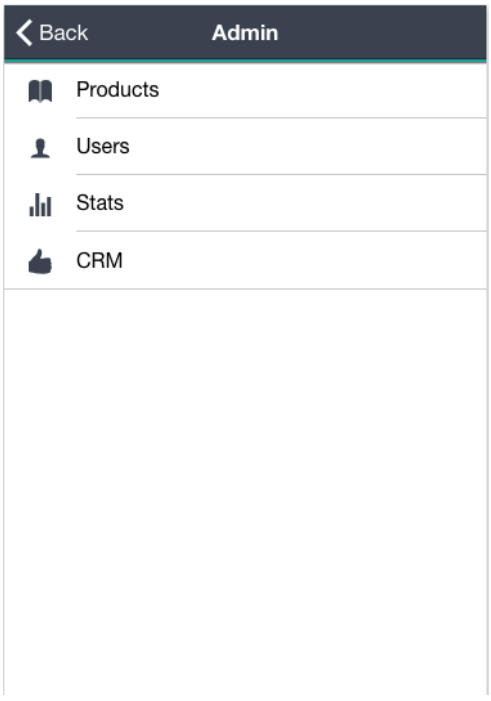


Ilustración 53 - Prototipo pantalla administración

Ilustración 54 - Pantalla final administración

Productos

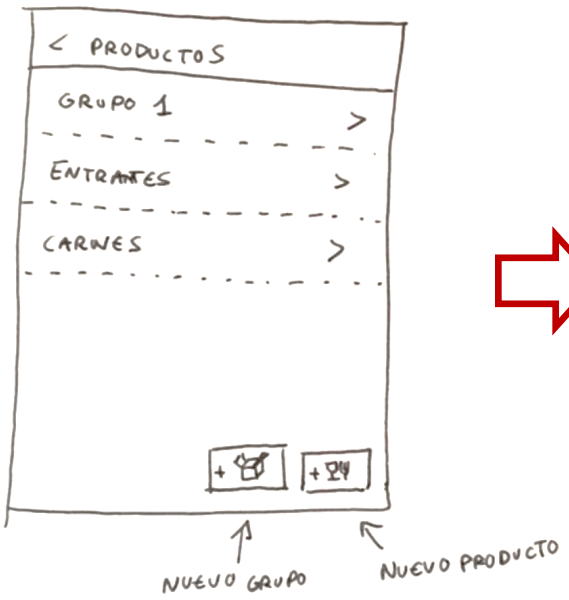


Ilustración 55 - Prototipo pantalla productos

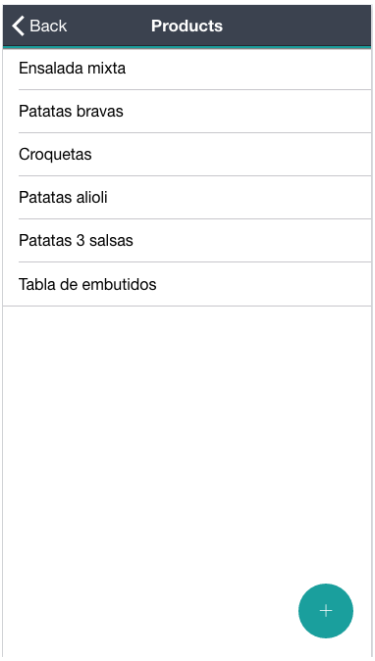


Ilustración 56 - Pantalla final productos

Detalle producto

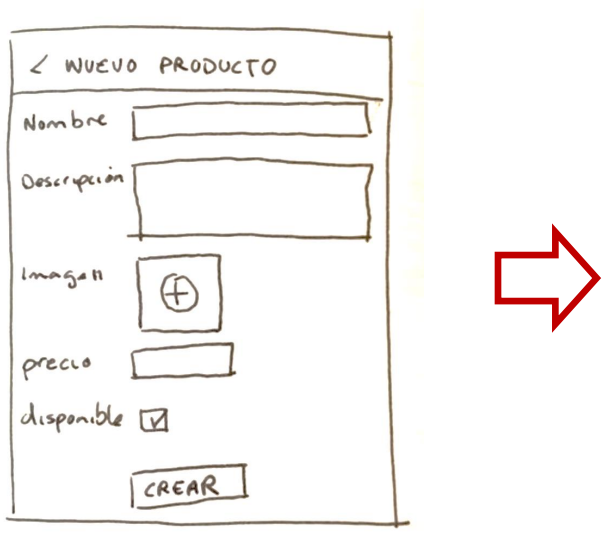


Ilustración 58 - Prototipo pantalla detalle producto

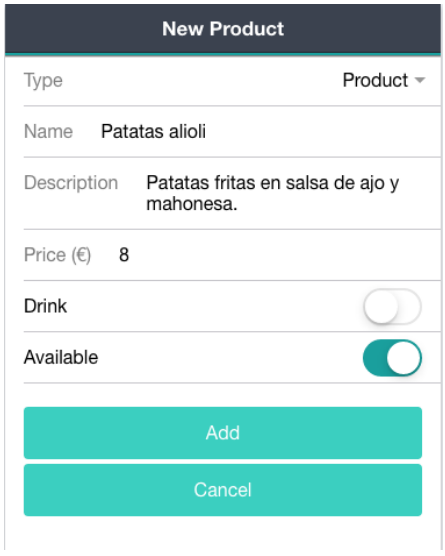


Ilustración 57 - Pantalla final detalle producto

Usuarios

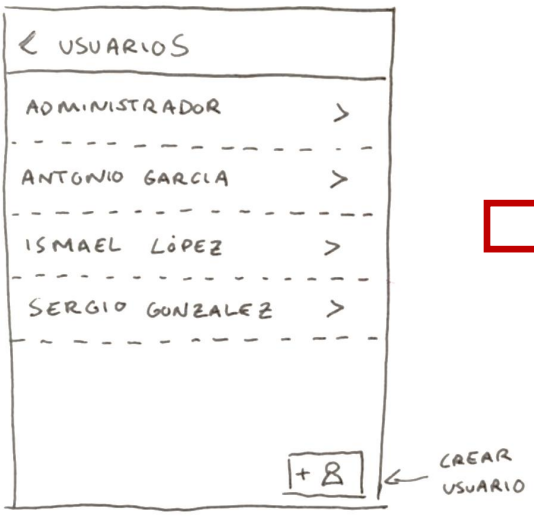


Ilustración 60 - Prototipo pantalla usuarios

Ilustración 59 - Pantalla final usuarios

Detalle usuario

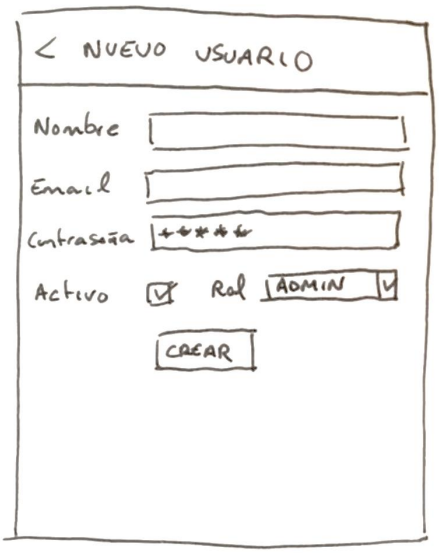


Ilustración 62 - Prototipo detalle usuario

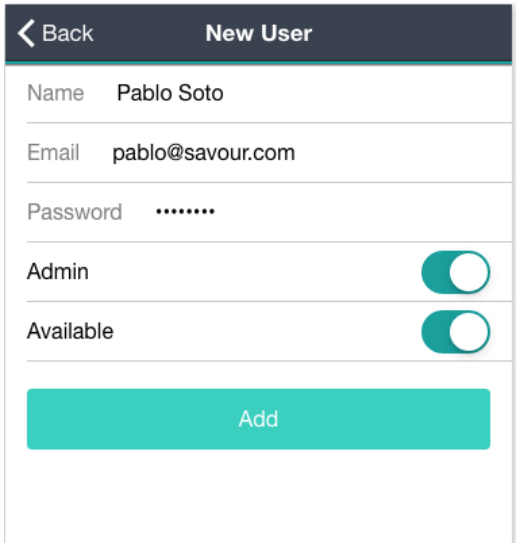


Ilustración 61 - Pantalla final detalle usuario

Estadísticas

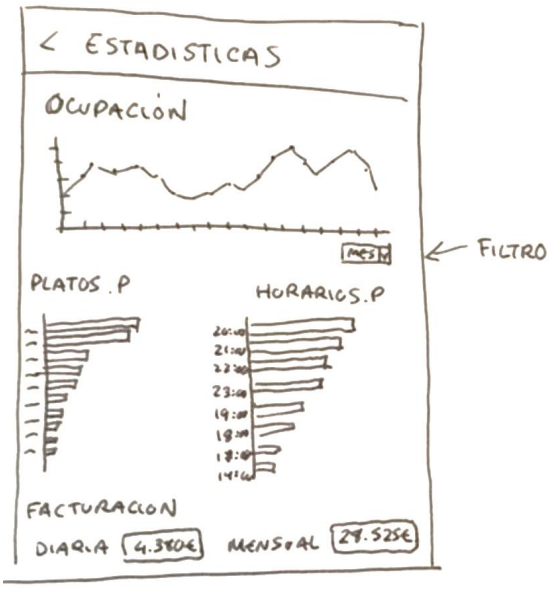


Ilustración 64 - Prototipo pantalla estadísticas

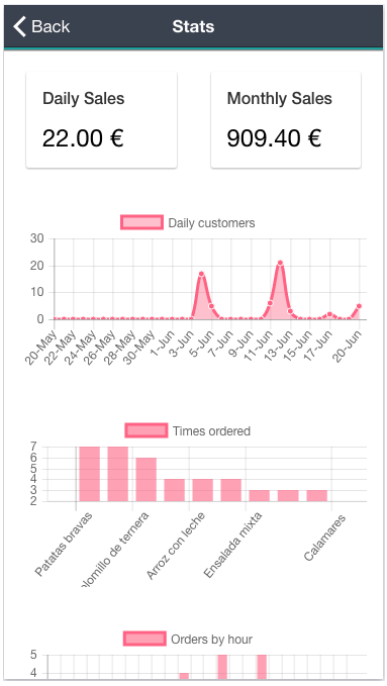


Ilustración 63 - Pantalla final estadísticas

Fidelización

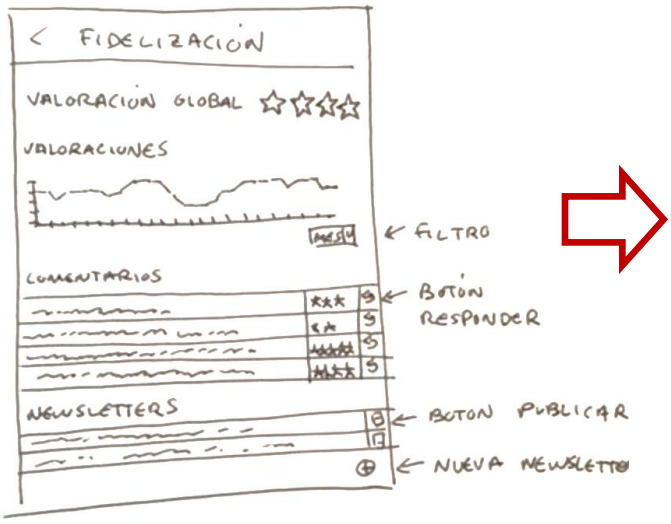


Ilustración 66 - Prototipo pantalla CRM

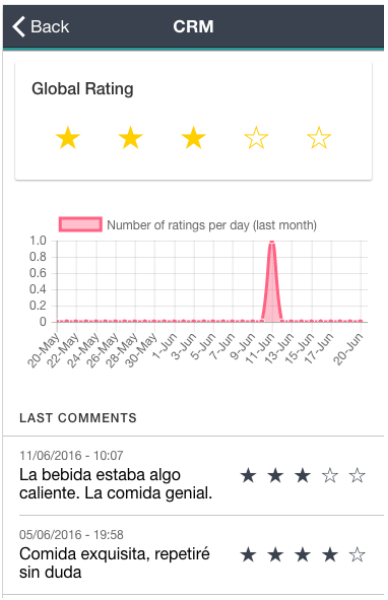
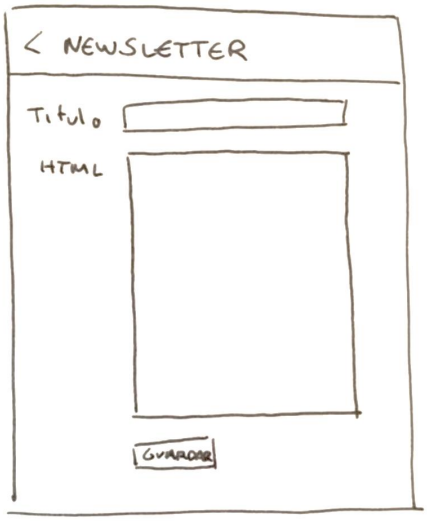


Ilustración 65 - Pantalla final CRM

Newsletter



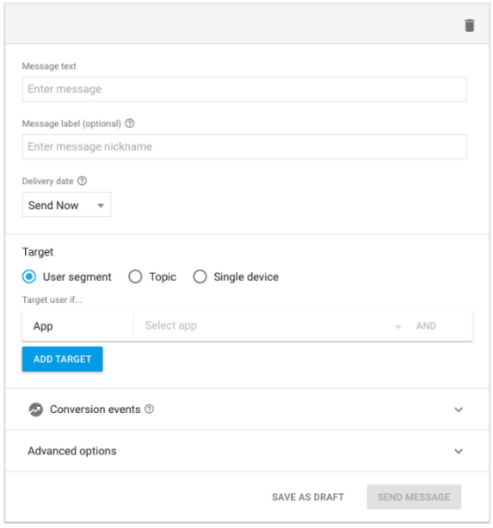


Ilustración 68 - Prototipo pantalla newsletter

Ilustración 67 - Firebase notifications

Nota: para esta última pantalla, se ha decidido utilizar el framework de notificaciones de Firebase pues es una herramienta magnífica para enviar notificaciones y mensajes segmentados por tipo de cliente.

5.5.2 Diseño de la interfaz para el subsistema aplicación móvil para el cliente

Identificación de la mesa

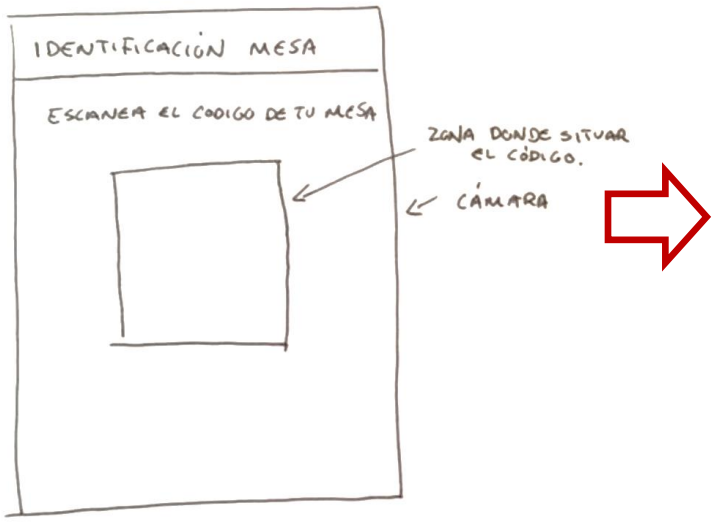


Ilustración 70 - Prototipo identificación de la mesa

Ilustración 69 - Pantalla final identificacion mesa

Carrito

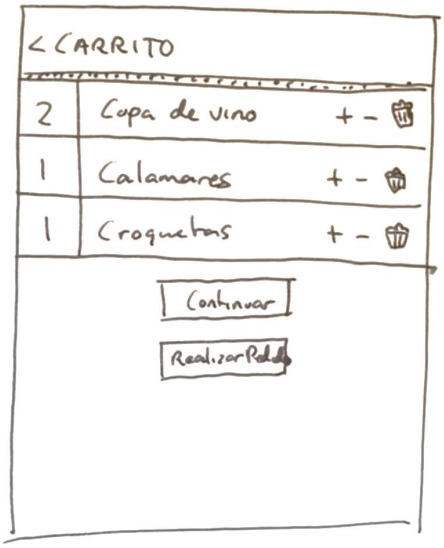


Ilustración 72 - Prototipo pantalla carrito

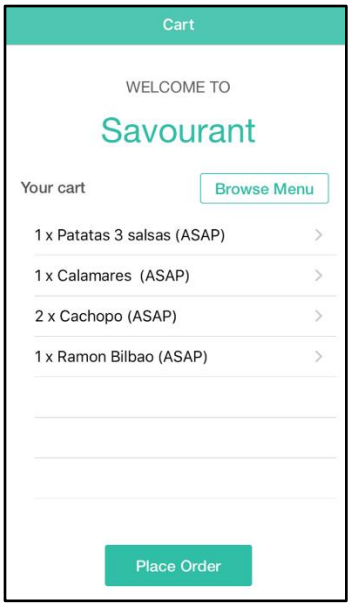


Ilustración 71 - Pantalla final carrito

Carta Virtual

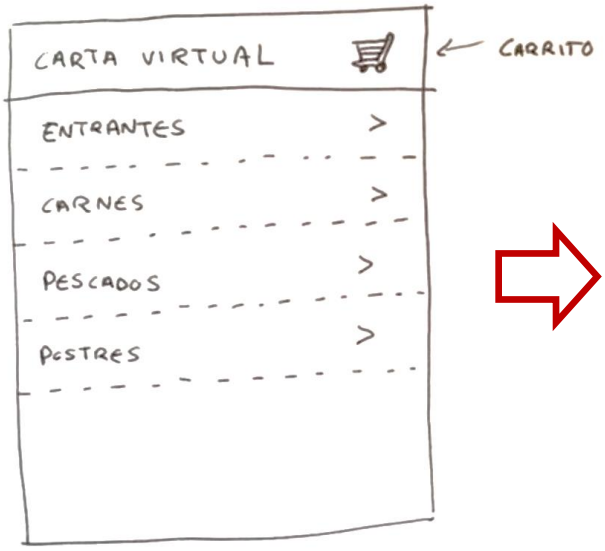


Ilustración 74 - Prototipo pantalla carta

Ilustración 73 - Pantalla final carta

Estado

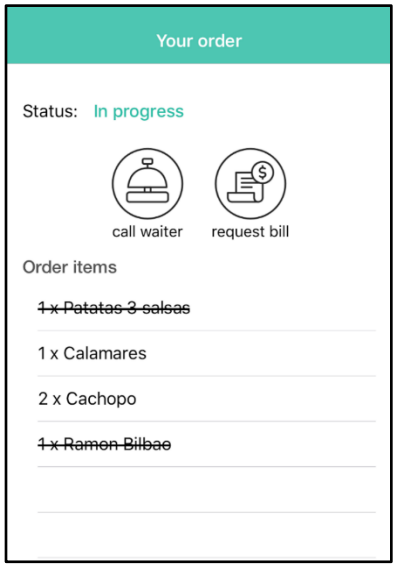


Ilustración 76 - Prototipo pantalla estado

Ilustración 75 – Pantalla final estado

Valoración

Ilustración 78 - Prototipo pantalla valoración

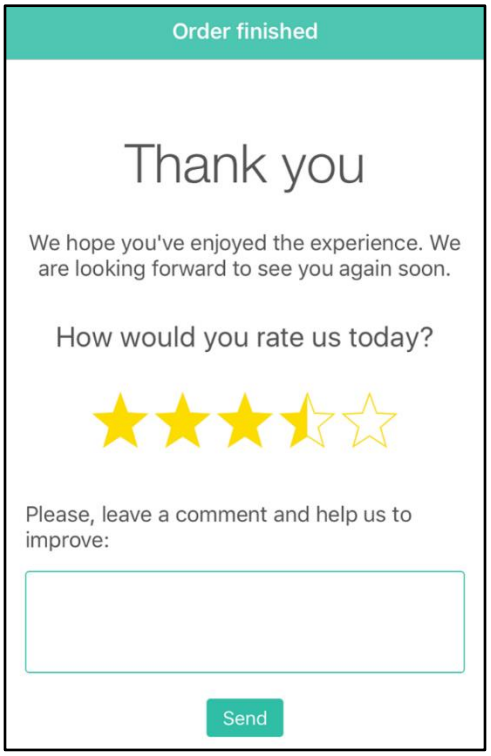


Ilustración 77- Pantalla final valoración

5.5.3 Trazabilidad Casos de Uso – Pantallas finales de la aplicación

	CU 1	CU 2	CU 3	CU 4	CU 5	CU 6	CU 7	CU 8	CU 9	CU10	CU11	CU12	CU13	CU14	CU15	CU16	CU17	CU18
Login	X																	
Selección vista		X	X															
Vista de barra				X	X													
Vista de sala						X												
Comanda							X	X										
Vista de cocina										X	X							
Cobro									X									
Vista de administración												X						
Productos													X					
Detalle producto													X					
Usuarios														X				
Detalle usuario														X				
Estadísticas															X			
Fidelización																X		
Newsletter																	X	X

Capítulo 6. Implementación del Sistema

6.1 Estándares y Normas Seguidos

Para el desarrollo de la aplicación web, se han seguido las convenciones de código oficiales de Microsoft para TypeScript descritas en:

https://github.com/Microsoft/TypeScript/wiki/Coding-guidelines

A su vez, se ha prestado especial atención a las guías de estilo para Angular2, sobre todo en cuanto a la organización del proyecto y código se refiere, disponibles en:

https://angular.io/docs/ts/latest/guide/style-guide.html

En la aplicación móvil para los clientes, se han seguido las normas de estilo marcadas por Apple para Swift:

https://swift.org/documentation/api-design-guidelines/

Finalmente, para la estructura de los datos en Firebase, se utilizó como referencia la guía de buenas prácticas disponible en:

https://www.firebase.com/docs/web/guide/structuring-data.html

.

6.2 Lenguajes utilizados en el Desarrollo

A continuación, se muestra un listado de los lenguajes de programación más relevantes utilizados durante el desarrollo del proyecto.

6.2.1 Typescript

Es sin duda el lenguaje más utilizado durante el proyecto. Se trata de un Superset de Javascript desarrollado por Microsoft que incluye tipos, las últimas características y novedades de ECMAScript 6 y propuestas futuras de ECMAScript 7 como funciones asíncronas y decoradores, que aún no son soportadas por los navegadores, actualmente anclados en ECMAScript 5 [8].

De esta forma, podemos programar aplicaciones con las nuevas características del lenguaje para luego compilar el código a una versión Javascript compatible con ECMAScript3, actualmente soportado por prácticamente todos los navegadores modernos.

A continuación, se muestra un ejemplo de una clase con Typescript y su homónima compilada para ECMAScript3:

```
1 class Greeter {
2     greeting: string;
3     constructor(message: string) {
4         this.greeting = message;
5     }
6     greet() {
7         return "Hello, " + this.greeting;
8     }
9 }
```

Ilustración 79 – Clase con Typescript

```
1 var Greeter = (function () {
2     function Greeter(message) {
3         this.greeting = message;
4     }
5     Greeter.prototype.greet = function () {
6         return "Hello, " + this.greeting;
7     };
8     return Greeter;
9 }());
```

Ilustración 80 - Clase anterior compilada para ECMAScript3

6.2.2 Swift

Lenguaje de programación moderno y de propósito general creado por Apple en 2014 para reemplazar a su obsoleto Objective-C. Es el lenguaje utilizado en la progración de la aplicación móvil IOS para los clientes.

Ilustración 81 - Logo oficial de Swift

Swift fue desarrollado basándose en 3 premisas básicas. La **seguridad**, pese a poder declarar variables sin tipo, el compilador es lo suficientemente inteligente como para comprobar las asignaciones. También los tipos opcionales aportan seguridad al evitar la utilización de valores nulos y la necesidad de forzar explícitamente su desempaquetado (unwrap).

Por otro lado, debía ejecutarse con la misma **rapidez** que su antecesor, y muy similar a lenguajes como C y C++ algo que en los lenguajes modernos es poco usual.

Finalmente, y probablemente el gran inconveniente de Objective-C, la sintaxis y **expresividad** del lenguaje debe manifestarse de forma natural a lo que espera el programador [13].

6.2.3 Sass

De forma similar a Typescript, se trata de un Superset en este caso sobre CSS. Surgió como necesidad para evitar que los proyectos con muchas líneas de código CSS se vuelvan tan complejos y difíciles de mantener.

Es totalmente compatible con CSS, por lo que puede utilizarse en hojas de estilos ya existentes para incluir sus nuevas características y finalmente ser compilados de nuevo a CSS plano entendible por el navegador.

Sus características más importantes son:

- **Variables**: Permite asignar cualquier valor a una variable para poder ser reutilizada. Por ejemplo un color primario ($primary: '#333').

- **Anidamiento**: Ya no es necesario repetir los nombres de los contenedores padres para varias de sus reglas hijas. Basta con anidarlas dentro del ámbito del padre.

- **Parciales**: Importación de otros ficheros Sass, sin que sean compilados a CSS.
- **Importaciones**: Actualmente la importación con CSS3 requiere una nueva petición HTTP. SASS, mergea los ficheros en un único resultante en tiempo de compilación para mejorar la eficiencia, manteniéndolos separados para el desarrollo y modularización.
- **Mixins**: Algo similar a las funciones en otros lenguajes de programación.
- **Herencia**: Permite incorporar los estilos de otras reglas ya definidas sin necesidad de volver a escribirlas.
- **Operadores**: Es posible realizar operaciones aritméticas con anchos y altos de elementos u otras medidas numéricas.

6.2.4 Otros

Durante el desarrollo del proyecto también se han utilizado otros lenguajes como:

- **HTML**: Para la maquetación estática de las vistas en la aplicación.
- **JSON**: Como formato de intercambio de datos con el backend.

6.3 Herramientas, Programas y Librerías Usados

6.3.1 Herramientas, programas y librerías para el desarrollo de la aplicación

IONIC Framework

Es el framework base sobre el que se ha desarrollado la aplicación web. IONIC proporciona una serie de componentes web de estilos muy similares a los componentes nativos de Android e Ios, y material design para la web, que permite desarrollar aplicaciones adaptables con tecnologías web para posteriormente ejecutarlas en el navegador o exportarlas como aplicaciones móviles .

Ilustración 82 – Logo de IONIC Framework

Utiliza Angular para el binding entre la capa de datos y las vistas y Cordova para la exportación a aplicaciones móviles. Internamente dichas aplicaciones despliegan el código del proyecto web sobre un webview.

Angular 2

Angular es un framework Javascript de código abierto, mantenido por Google y utilizado para el desarrollo de aplicaciones web de una sola página. Para el presente proyecto se ha utilizado la versión 2 que actualmente se encuentra en fase de RC (Release Candidate) y que no es compatible con su versión anterior, cambiando gran parte de su núcleo y características.

Ilustración 83 - Logo de Angular

Aunque inicialmente nació como un framework MVC (Model-View-Controller), sus numerosas refactorizaciones y cambios en sus características lo han acabado por resultar más bien un MVVM (Model-View-ViewModel), o como sus propios desarrolladores definen, un MVW (Model-View-Whatever), por su capacidad de adaptarse a lo que necesite el programador.

Para el desarrollo de este proyecto se ha utilizado más bien el patrón MVVM, por la naturaleza reactiva de Firebase, que, junto con los databindings de Angular, hacen una gran combinación [9][7].

Firebase

Hasta hace poco se conocía como una base de datos NoSQL alojada en la nube y con sincronización en tiempo real. Recientemente, tras haber sido adquirida por Google, se ha convertido en una suite de productos de desarrollo, sin embargo, para este proyecto sólo se utiliza su característica más llamativa, que sigue siendo la base de datos, y que constituye el back-end de la aplicación.

La elección de Firebase como backend no es trivial. Es su característica de sincronización reactiva en tiempo real la que lo hace perfecta para esta aplicación. Gracias a el, los usuarios de la aplicación, que muchas veces simplemente utilizarán la aplicación para consulta como en cocina, o en barra, no necesitarán refrescar el navegador para ver sus datos actualizados.

Para ello, Firebase utiliza Websockets, manteniendo una conexión activa para cada usuario conectado, e intercambiando datos en tiempo real, evitando las latencias de peticiones recurrentes HTTP o AJAX. [10][11][12]

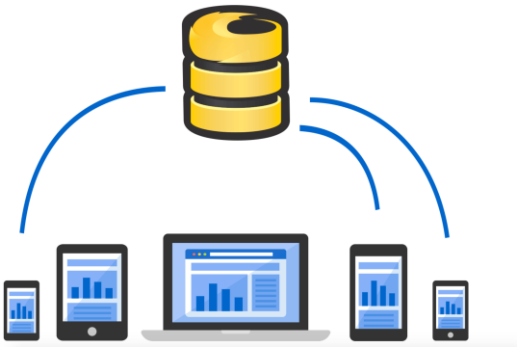


Ilustración 84 - Sincronización de Firebase con múltiples dispositivos

Por otro lado, está preparada para escalar automáticamente y para funcionar offline, otra característica interesante para que si se producen cortes en el servicio de internet no provoquen pérdidas en los datos.

Webstorm

IDE de Jetbrains para el desarrollo de proyectos web. Se ha utilizado por sus numerosas utilidades para el autocompletado de tipos, edición y mantenimiento de código, automatización, etc.

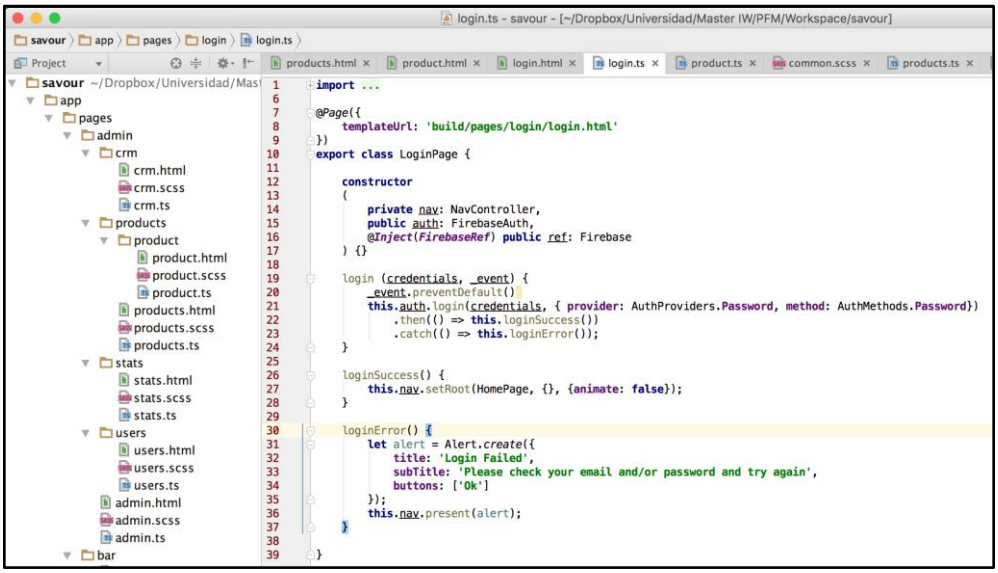


Ilustración 85 - WebStorm durante el desarrollo del proyecto

XCode

Se trata del entorno de desarrollo de Apple para el desarrollo de Aplicaciones para IOS y OSX. Fue utilizado para el desarrollo de la aplicación nativa para el cliente, utilizando el lenguaje de programación Swift.

Cuenta con simuladores para todos los dispositivos móviles Apple, aunque también es posible utilizar los dispositivos físicos, que proporcionan mayor rendimiento en la ejecución.

6.3.2 Herramientas auxiliares utilizadas en el proyecto

Microsoft Office 2013

Se trata de un paquete ofimático del que se han utilizado las siguientes aplicaciones:

- **Microsoft Word 2013**: Herramienta para el procesamiento de textos. Ha sido la herramienta utilizada para crear y dar forma a este documento.

- **Microsoft Excel 2013:** Es una herramienta destinada al manejo de hojas de cálculo. Se ha utilizado para realizar el presupuesto del proyecto y algunos diagramas estadísticos incluidos en este documento.

- **Microsoft Project 2013:** Herramienta para la planificación y gestión de proyectos. Permite mantener un control sobre el trabajo, la programación y las finanzas. Ha sido utilizada para planificar el desarrollo de este proyecto.

Lucidchart

Se trata de una herramienta web de modelado y diseño de software que permite realizar diagramas UML, mind maps, bocetos organizativos y muchos otros tipos de diagramas con la peculiaridad de que permite realizar los diseños a través del navegador y almacenar los documentos en la nube, que posteriormente pueden ser compartidos o utilizados en entornos colaborativos.

Lucidchart es utilizado en ciertas secciones del proyecto en las que es necesario crear otro tipo de diagramas desvinculados de UML, o que requieran cierta flexibilidad en el diseño que Enterprise Architect no permita.

Se puede acceder a este servicio a través de la página web de Lucidchart (www.lucidchart.com).

6.4 Problemas encontrados

Versiones Beta de IONIC y Angular

Cuando comencé el proyecto, y tras evaluar las posibles alternativas de lenguajes y frameworks para el desarrollo, tenía clara la elección de utilizar IONIC. En ese momento existían dos versiones disponibles, IONIC 1.3, que utilizaba internamente Angular en su primera versión, ambas estables, o una nueva versión IONIC 2 (Beta 5) que utilizaba internamente Angular 2 (beta 15).

Ya que se trataba de un nuevo proyecto, decidí lanzarme a utilizar las nuevas versiones a pesar de no ser muy estables. Este hecho provocó que a lo largo del desarrollo tuviese que actualizar varias veces las librerías y arreglar muchos de los cambios no compatibles que se introducían con las nuevas versiones.

Finalmente, se evolucionó IONIC desde su beta 5 hasta su beta 9 y Angular2 desde la beta 15 hasta la Release Candidate 2. Probablemente, de comenzar un nuevo proyecto en este momento haría lo mismo, aún más ahora que la versión de Angular se aproxima mucho a su lanzamiento final, pero esta vez si lo tendría en cuenta de cara a una posible desviación en la planificación.

Firebase sin actualizaciones realtime

Uno de los motivos principales de elegir Firebase como backend fue su característica de base de datos en tiempo real, que lo hacía perfecto para combinar con la programación reactiva de Angular2.

En las primeras semanas de desarrollo siempre funcionó a la perfección, pero llegó un momento en el que misteriosamente dejó de responder a los cambios pasados varios segundos. Después de analizar la situación durante mucho tiempo, llegué a la conclusión de que el problema sólo aparecía cuando se utilizaba Safari como navegador, o al exportar la app a dispositivos móviles.

Finalmente, rastreando la red en búsqueda de una respuesta, conseguí resolverlo introduciendo la siguiente línea en el fichero index.html:

```
<meta http-equiv="Content-Security-Policy" content="default-src *; style-src 'self'
'unsafe-inline' *; script-src 'self' 'unsafe-inline' 'unsafe-eval' *">
```

Básicamente, se encarga de decir al navegador que puede confiar (white-list) en los scripts de la página. Al parecer, safari y los navegadores móviles (a través de Cordova) bloqueaban los websockets de firebase una vez finalizaban las primeras llamadas.

Validación de formularios con Angular2

Este problema viene derivado del primer problema descrito, pero he decidido tratarlo por separado pues ha supuesto bastantes cambios no esperados.

El día 3 de Junio de 2016, ante el lanzamiento inminente de Angular RC2, se publica un artículo [14] con una propuesta de cambios en la estrategia de creación de formularios y validación, no compatible con la versión anterior, por lo que rompe el código utilizado hasta la fecha para la aplicación. A continuación, se muestra uno de los warnings recibidos desde la consola javascript:

```
⚠ *It looks like you're using the old forms module. This will be opt-in in the next RC, and
will eventually be removed in favor of the new forms module. For more information, see:
https://docs.google.com/document/u/1/d/1RIezQqE4aEhBRmArIAS1mRIZtWFf6JxN_7B4meyWK0Y/pub
```

Los principales problemas venían derivados de cuestiones de legibilidad, nomenclatura y al tratar de compatibilizar elementos entre Angular1 y Angular2 que realmente no eran compatibles al 100% y que podían causar confusión a los desarrolladores que migrasen desde la antigua versión.

Ante estos cambios, fue necesario realizar una reestructuración total de los formularios, para hacerlos compatibles con las nuevas versiones.

Capítulo 7. Desarrollo de las Pruebas de Aceptación

7.1 Primer pase de la batería de pruebas

A continuación se muestra una tabla con los resultados de la evaluación de las pruebas diseñadas en el apartado 4.5, categorizadas según los casos de uso en cada uno de ellos.

Caso de Uso 1: Autenticarse

Caso de prueba	Resultado	Observaciones
1	✓	
2	✓	
3	Fallida	Se permite el acceso con el usuario desactivado.

Caso de Uso 2: Salir

Caso de prueba	Resultado	Observaciones
4	✓	
5	✓	
6	Fallida	Sólo nos reenvía al login desde las páginas internas de administración (usuarios, stats,…) pero no desde la administración en sí.

Caso de Uso 3: Seleccionar vista

Caso de prueba	Resultado	Observaciones
7	✓	
8	✓	
9	✓	
10	Fallida	La opción es visible a usuarios no administradores.

Caso de Uso 4: Vista de barra

Caso de prueba	Resultado	Observaciones
11	Fallida	Los pedidos no aparecen bien ordenados por prioridad.

Caso de Uso 5: Marcar bebida preparada

Caso de prueba	Resultado	Observaciones
12	✓	

Caso de Uso 6: Vista de sala

Caso de prueba	Resultado	Observaciones
13	✓	

Caso de Uso 7: Gestión de comanda

Caso de prueba	Resultado	Observaciones
14	Fallida	Permite crear una comanda sin ningún producto.
15	Fallida	Permite editar la comanda y dejarla sin ningún producto.
16	✓	

Caso de Uso 8: Marcar pasos de mesa

Caso de prueba	Resultado	Observaciones
17	✓	

Caso de Uso 9: Cobro de factura

Caso de prueba	Resultado	Observaciones
18	Fallida	Al eliminar el último producto de una orden, esta no se marca como finalizada y por tanto no es posible realizar el cobro.
19	✓	

Caso de Uso 10: Vista de cocina

Caso de prueba	Resultado	Observaciones
20	✓	

Caso de Uso 11: Marcar comida como preparada

Caso de prueba	Resultado	Observaciones
21	✓	

Caso de Uso 12: Vista de administración

Caso de prueba	Resultado	Observaciones
22	Fallida	Permite el acceso a usuarios no administradores.

Caso de Uso 13: Gestión de productos

Caso de prueba	Resultado	Observaciones
23	✓	
24	✓	
25	✓	
26	✓	
27	Fallida	Hay un bug con el gesto "swipe" que muestra la opción de eliminación y es imposible activar dicha opción.

Caso de Uso 14: Gestión de usuarios

Caso de prueba	Resultado	Observaciones
28	✓	
29	Fallida	Falla la edición de un usuario. La pantalla se queda bloqueada.
30	Fallida	Falla la desactivación por violar una restricción de base de datos incorrecta.

Caso de Uso 15: Estadísticas

Caso de prueba	Resultado	Observaciones
31	✓	
32	✓	
33	✓	
34	✓	

Caso de Uso 16: CRM

Caso de prueba	Resultado	Observaciones
35	✓	
36	Fallida	Sólo muestra valoraciones 1 día concreto, aunque haya valoraciones otros días del mes.

Caso de Uso 17: Comentarios de clientes

Caso de prueba	Resultado	Observaciones
37	✓	

Caso de Uso 18: Publicación de newsletter

Caso de prueba	Resultado	Observaciones
38	✓	

7.2 Segundo pase de la batería de pruebas

Una vez resueltos los fallos encontrados en el primer pase de pruebas, se realiza otro pase de pruebas con los siguientes resultados.

Caso de Uso 1: Autenticarse

Caso de prueba	Resultado	Observaciones
1	✓	
2	✓	
3	✓	

Caso de Uso 2: Salir

Caso de prueba	Resultado	Observaciones
4	✓	
5	✓	
6	✓	

Caso de Uso 3: Seleccionar vista

Caso de prueba	Resultado	Observaciones
7	✓	
8	✓	
9	✓	
10	✓	

Caso de Uso 4: Vista de barra

Caso de prueba	Resultado	Observaciones
11	✓	

Caso de Uso 5: Marcar bebida preparada

Caso de prueba	Resultado	Observaciones
12	✓	

Caso de Uso 6: Vista de sala

Caso de prueba	Resultado	Observaciones
13	✓	

Caso de Uso 7: Gestión de comanda

Caso de prueba	Resultado	Observaciones
14	✓	
15	✓	
16	✓	

Caso de Uso 8: Marcar pasos de mesa

Caso de prueba	Resultado	Observaciones
17	✓	

Caso de Uso 9: Cobro de factura

Caso de prueba	Resultado	Observaciones
18	✓	
19	✓	

Caso de Uso 10: Vista de cocina

Caso de prueba	Resultado	Observaciones
20	✓	

Caso de Uso 11: Marcar comida como preparada

Caso de prueba	Resultado	Observaciones
21	✓	

Caso de Uso 12: Vista de administración

Caso de prueba	Resultado	Observaciones
22	✓	

Caso de Uso 13: Gestión de productos

Caso de prueba	Resultado	Observaciones
23	✓	
24	✓	
25	✓	
26	✓	
27	✓	

Caso de Uso 14: Gestión de usuarios

Caso de prueba	Resultado	Observaciones
28	✓	
29	✓	
30	✓	

Caso de Uso 15: Estadísticas

Caso de prueba	Resultado	Observaciones
31	✓	
32	✓	
33	✓	
34	✓	

Caso de Uso 16: CRM

Caso de prueba	Resultado	Observaciones
35	✓	
36	✓	

Caso de Uso 17: Comentarios de clientes

Caso de prueba	Resultado	Observaciones
37	✓	

Caso de Uso 18: Publicación de newsletter

Caso de prueba	Resultado	Observaciones
38	✓	

Capítulo 8. Conclusiones y Ampliaciones

8.1 Conclusiones

Finalizado el proyecto, tengo que decir que me siento muy satisfecho con el trabajo realizado y el resultado obtenido. Creo que el tipo de proyecto encajaba perfectamente con lo que supone un trabajo final del Máster en Ingeniería Web en su rama profesional, tanto por el contenido como por las tecnologías empleadas.

Durante su desarrollo se realizaron 2 reuniones, al comienzo y al final, con personas de amplia experiencia en el sector de la hostelería, y que sirvieron de gran ayuda, aportando mucho conocimiento, influyendo en la toma de decisiones y permitiendo elaborar un producto con sentido.

El proyecto cumple todos los objetivos propuestos inicialmente, excepto el de pagos virtuales, que se desestimó y se logró pivotar a raíz de las reuniones anteriores, pues carecía de sentido tal y como se había planteado inicialmente.

La elección de las tecnologías resultó clave, ya que a día de hoy no hay demasiadas alternativas en el ámbito web que permitan la creación de aplicaciones adaptables a dispositivos móviles, dinámicas y con comportamiento realtime de una forma sencilla. Tanto Firebase como Ionic ayudaron mucho en este aspecto y permitieron realizar el proyecto en tiempos mucho más cortos que de haber confiado en otras herramientas o tecnologías.

También hubo momentos complicados. Aparecieron varios problemas no esperados y el tiempo disponible era muy limitado, pues el desarrollo del proyecto fue compaginado con un trabajo de responsabilidad a jornada completa y en muchas ocasiones resultaba imposible dedicar el tiempo suficiente, lo que provocó en gran medida las numerosas desviaciones en la planificación.

Finalmente, fruto del trabajo, la constancia y gracias al apoyo de mis personas más allegadas, logro finalizar el proyecto en tiempo y forma, lo que ha supuesto la superación de un nuevo reto para mí, y del que me siento muy orgulloso.

8.2 Posibles ampliaciones

A continuación, se comentan una serie de ampliaciones, no previstas inicialmente y que han ido surgiendo durante el desarrollo del proyecto:

- **Actualización a Firebase 3**: Durante el Google IO de 2016 (finales de mayo), se anunció una nueva versión de la librería de Firebase con cambios y mejoras importantes, pero que no fue posible incluir en el proyecto por incompatibilidades con otras librerías como AngularFire, que venían siendo utilizadas y que aún no soportaban dicha versión. Es muy probable que durante el mes de Julio de 2016 se actualizen dichas librerías y pueda llevarse a cabo la migración.

- **Mejora de las estadísticas**: Introducir nuevas estadísticas como porcentajes de ocupación, tiempo medio por comanda, facturación media por comensal, etc. Además de incluir filtros a las existentes.

- **Validación exahustiva**: Pese a que se está realizando validación tanto en el cliente como en el servidor, hay muchos elementos que se pueden mejorar, tal y como se comentó en la sección 5.4.3.

- **Fotografías para los productos**: Sobretodo de cara a la exploración de la carta por parte de los clientes, sería interesante poder asociar fotos a los productos desde la administración, que luego se visualizasen en la aplicación.

- **Integración de Firebase Notifications en el panel de administración**: Durante el desarrollo del proyecto se deció utilizar firebase notifications para la publicación de mensajes para los usuarios. El problema es que para ello es necesario utilizar una web externa, y sería de gran utilidad que esta funcionalidad estuviese integrada en el panel de administración. Por el momento Firebase no proporciona ningún mecanismo de integración con las notificaciones, pero se prevee que lo incluya en el futuro.

Capítulo 9. Referencias Bibliográficas

9.1 Libros y Artículos

- **Dr. Axel Rauschmayer. "Speaking Javascript", Editorial O'Reilly. ISBN-13: 978-1449365035**:

  Ha sido uno de los principales libros de referencia para el desarrollo del proyecto, probablemente uno de los más actualizados sobre el lenguaje Javascript.

- **David Flanagan. "Javascript. The definitive guide, 6th edition", Editorial O'Reilly. ISBN-13: 978-059680552**:

  Libro adquirido durante el Máster en Ingenería Web como recomendación. La sexta edición trata todos los aspectos del lenguaje Javascript hasta ECMAScript 5.

- **Nathan Rozentals. "Mastering TypeScript", Editorial Packthub. ISBN: 1784399663**:

  Libro ojeado durante el desarrollo, junto a la documentación oficial de Microsoft. Uno de los pocos libros sobre Typescript del mercado.

- **Julien Lange. "Swift 2 Design patterns", Editorial Packthub. ISBN-13: 978-1-78588-761-1**:

  Patrones de diseño orientados al desarrollo de juegos para Swift en su version 2. Se han extraido algunos recursos sobre estructura y organización del código.

- **VV.AA. "Design Patterns: Elements of Reusable Object-Oriented Software", Editorial Addison-Wesley. ISBN-13: 978-02-016-3361-0***:*

  Todo un clásico sobre patrones de diseño tambien conocido como GoF (Gang of Four), siempre lo utilizo como referencia en todos mis proyectos.

9.2 Referencias en Internet

[1] Comparativa de soluciones de punto de venta para bares y restaurantes. http://www.softwareadvice.com/retail/bar-pos-software-comparison/

[2] Mise en place (Organización de los procesos de sala). http://www.gestionrestaurantes.com/organizacion-de-los-procesos-de-sala-i-la-puesta-a-punto-o-mise-en-place-en-la-sala/

[3] Mecanica en el servicio de sala (Organización de los proceso de sala). http://www.gestionrestaurantes.com/organizacion-de-los-procesos-de-sala-ii-la-mecanica-en-el-servicio-de-sala/

[4] Mise en place (Wikipedia). https://es.wikipedia.org/wiki/Mise_en_place

[5] Presentación de la comanda (Escuela de hostelería de Aragón). http://www.escuelahosteleria.org/portal/recetas/materiales/3hvajL1St.pdf

[6] Servicios de restauración del gobierno de canarias, la comanda. http://www3.gobiernodecanarias.org/medusa/ecoblog/avennav/la-comanda/

[7] Guía rápida de Angular 2 (ionic team). http://learnangular2.com/

[8] Web official de typescript. https://www.typescriptlang.org

[9] AngularFire2. Wrapper sobre Firebase incluyendo extensiones reactivas para angular2. https://angularfire2.com/api/

[10] The key to Firebase Security. Google IO 2016. https://www.youtube.com/watch?v=PUBnlbjZFAI

[11] Deep dive into the realtime database (Firebase, Google IO 2016). https://www.youtube.com/watch?v=cYinms8LurA

[12] Arrays in Firebase, Best practices. https://firebase.googleblog.com/2014/04/best-practices-arrays-in-firebase.html

[13] The Swift programming language (Apple reference guide). https://developer.apple.com/library/prerelease/content/documentation/Swift/Conceptual/Swift_Programming_Language

[14] Forms change proposal. Angular2 team (June 3, 2016). https://docs.google.com/document/u/1/d/1RIezQqE4aEhBRmArIAS1mRIZtWFf6JxN_7B4meyWK0Y/pub

Capítulo 10. Apéndices

10.1 Diario de desarrollo del proyecto

Fecha	Descripción de la tarea	Horas	Tipo
02/12/2014	Planificación inicial PFM	3	Documentación
02/01/2015	Presupuesto	3	Documentación
23/01/2015	Anteproyecto	1,5	Documentación
23/01/2015	Tutoría con director de proyecto	2	Tutoría
28/02/2015	Mockup pantallas	3	Estudios previos
01/03/2015	Reunión con representante de sector hostelería	3	Estudios previos
08/03/2015	Justificación del proyecto	1,5	Documentación
20/03/2015	Estudio de la situación actual – Parte 1	4	Estudios previos
25/06/2015	Focus Group #1	4	Estudios previos
15/01/2016	Estudio de la situación actual – Parte 2	6	Estudios previos
17/01/2016	Objetivos del proyecto	1	Documentación
17/01/2016	Estudio IONIC framework, Apache Cordova	8	Formación
06/02/2016	Aspectos teóricos	8	Documentación
20/02/2016	Unión documentación	4	Documentación
27/02/2016	Identificación subsistemas y requisitos 1	6	Análisis
28/02/2016	Identificación subsistemas y requisitos 2	5	Análisis
13/03/2016	Casos de uso 1	5	Análisis
19/03/2016	Casos de uso 2 – Diagramas y trazabilidad	7	Análisis
10/04/2016	Modelo de dominio – diagrama y descripción clases	5	Análisis
17/04/2016	Navegabilidad y prototipos de pantallas en app web	7	Análisis
18/04/2016	Prototipos de pantallas en app móvil y trazabilidad con C.U	5	Análisis
01/05/2016	Demo pantalla de login	5	Implementación
07/05/2016	Migración IONIC2 + Angular2	3	Implementación
08/05/2016	Typescript	4	Formación
14/05/2016	Firebase – real time database	5	Formación
15/05/2016	Pruebas con proyecto demo Ionic + AngularFire	4	Formación
20/05/2016	Diagrama de despliegue	1	Diseño
22/05/2016	Pantalla de login funcional	2	Implementación
23/05/2016	Pangalla de home con logout y estilos	4	Implementación
24/05/2016	Revisión de la planificación	2	Documentación
28/05/2016	Submenú de administración y listado de productos	6	Implementación
29/05/2016	Mantenimiento de productos y grupos	8	Implementación
30(05/2016	Tutoría con director de proyecto	2	Tutoría
30/05/2016	Estándares y normas	1	Documentación
30/05/2016	Lenguajes utilizados	2	Documentación
30/05/2016	Herramientas programas y librerías	2	Documentación
02/06/2016	Modelo de datos comanda	2	Implementación
04/06/2016	Implementación pantalla de comanda	7	Implementación

04/06/2016	Implementación pantalla de sala	4	Implementación
05/06/2016	Implementación pantalla de sala	2	Implementación
05/06/2016	Implementación pantalla de barra	5	Implementación
05/06/2016	Implementación pantalla de cocina	1	Implementación
05/06/2016	Implementación cobro de comanda	3	Implementación
07/06/2016	Descripción diagrama de despliegue	1	Diseño
11/06/2016	Mantenimiento de usuarios	7	Implementación
11/06/2016	Estadísitcas. Instalación angular-charts	4	Implementación
12/06/2016	Pantalla estadísticas (facturación, ocupación, productos fav.)	8	Implementación
12/06/2016	Diagramas de secuencia	4	Diseño
14/06/2016	Solicitud de defensa	1	Documentación
14/06/2016	Pantalla CRM	2	Implementación
15/06/2016	Implementación servicios CRM	2	Implementación
18/06/2016	Comentarios CRM	3	Implementación
19/06/2016	Diseño de la interfaz (comparativa)	4	Diseño
20/06/2016	Arquitectura del sistema – Patrones de diseño	2	Diseño
20/06/2016	Arquitectura del sistema – Diagrama de componentes	2	Diseño
20/06/2016	Problemas encontrados	2	Documentación
20/06/2016	Cambios en la validación de formularios	4	Implementación
21/06/2016	Refactoring de clases y servicios	5	Implementación
21/06/2016	Esquema de la base de datos, seguridad, autentificación..	5	Diseño
22/06/2016	Corrección de bug creación de pedidos duplicados	3	Implementación
22/06/2016	Inclusión de focus group #1 en la documentación y preparación de focus group #2.	3	Documentación
22/06/2016	Pruebas aceptación (análisis)	4	Análisis
23/06/2016	Pase de pruebas de aceptación 1	3	Pruebas
23/06/2016	Corrección bugs pruebas	2	Pruebas
23/06/2016	Pase de pruebas aceptación 2	2	Pruebas
23/06/2016	Focus group 2	2	Documentación
23/06/2016	Inclusión focus group 2 en documentación	1	Documentación
24/06/2016	Estructura App iOS	2	Implementación
24/06/2016	Lectura de código QR e identificación	3	Implementación
24/06/2016	Integración Firebase y listado de productos	5	Implementación
25/06/2016	Creación de comandas Ios	4	Implementación
25/06/2016	Pantalla de estado autoactulizable	4	Implementación
25/06/2016	Pantalla de valoraciones	2	Implementación
26/06/2016	Abstract, descripción idea, planificación real, comparativa planificaciones, referencias, inclusión código fuente, revisión general.	10	Documentación
27/06/2016	Revisión	5	Documentación
27/06/2016	Pruebas aplicaciones	5	Pruebas

10.2 Reuniones con personal del sector

10.2.1 Focus Group 1

Fecha: 25 de Junio de 2015

Persona Colaboradora: Jose Manuel García Fernández (extrabajador del sector, con experiencia durante más de 30 años en negocios de hostelería, por cuenta propia y ajena, desempeñando cargos de índole diversa).

Objetivos de la sesión

Puesto que se trata del primero contacto del proyecto con una persona con experiencia en el sector, y de forma previa al desarrollo del mismo, se establecieron los siguientes objetivos:

- **Describir de la idea**: transmitir de qué trata el proyecto, la motivación y sus beneficios esperados.
- **Recibir opiniones objetivas**: obtener feedback sincero para realizar los cambios necesarios en el desarrollo del proyecto.
- **Información sobre aspectos teóricos**: conseguir nociones básicas sobre elementos tan importantes como las comandas y sus fases.
- **Conocer la situación actual**: metodología, software existente, qué elementos resultan más útiles, sus costes asociados, etc.

Descripción de la idea

A continuación, se recoje la descripción de la idea inicial que se utilizó en la preparación previa de la reunión, y que finalmente se transmitió en su transcurso:

Se trata de una aplicación que permite sustituir o complementar las TPV tradicionales para la recepción de pedidos desde dispositivos móviles tanto por parte de los camareros y personal del propio negocio como por los clientes.

De esta forma, para realizar un nuevo pedido, sería necesario introducir el número de mesa, que los camareros podrían teclear a mano, o utilizar un código QR que identifique tanto mesa como local por parte de los clientes. Posteriormente, aparecería un listado de productos que podrían seleccionarse e ir añadiéndose a un carrito para finalmente procesar la orden.

La gran ventaja residirá en que una vez recibido un nuevo pedido, y de forma automática, este llegará tanto a una pantalla situada en la barra (solo bebida) como a otra pantalla situada en cocina (solo comida). Ambas pantallas se irán actualizando automáticamente según se vayan produciendo más órdenes, sin necesidad de intervención del personal, estando además los productos ordenados por prioridad.

Además, se incluiría la posibilidad de que los propios clientes evalúen su satisfación con los pedidos, en lugar de hacerlo en plataformas externas.

Los beneficios de una solución de este tipo serían múltiples, y podrían resumirse en los siguientes:

- *Los clientes no estarían obligados a esperar para realizar un nuevo pedido.*
- *El personal estaría más liberado al necesitar menor atención de los clientes.*
- *Ambos puntos anteriores repercutirían en beneficios para el negocio al acelerarse los procesos, que permitiría atender más clientes a lo largo de una sesión.*
- *Al tratarse de un proceso de pedidos informatizado, serían posible obtener el estado en tiempo real del negocio, stock, beneficios, etc.*
- *Conocer la opinión de los clientes es probablemente la mejor forma de hacer prosperar el negocio.*

Opiniones recibidas

Tras la explicación de la idea, Jose Manuel se mostró un poco escéptico. Había trabajado con software similar para PDAs y no había tenido demasiado éxito. Por otro lado, no acababa de entender en qué se diferenciaba la idea del software existente para realizar pedidos remotos.

Se explicó que, con esta solución, la información no sólo llegaba a un terminal en la barra, sino que también lo hacía a la **cocina donde él sí veía una necesidad importante**. Además, no se utilizarían PDAs por parte de los camareros sino **cualquier dispositivo móvil** con un navegador web, algo que valoró muy positivamente ya que las PDA con las que había trabajado daban muchos problemas con sus pantallas táctiles.

Posteriormente se trazaron varios bocetos de lo que podrían ser las pantallas de la aplicación, para que Jose Manuel ampliase su idea mental e identificase algún posible problema, detectando rápidamente uno de ellos, los **pasos de mesa**.

El problema residía en que no todos los elementos de una comanda tienen la misma prioridad. Si un cliente ordenaba un primer plato y un segundo, deberían prepararse dejando un espacio temporal entre ambos para evitar que se enfriase o se acumulase en cocina. Actualmente, y en la hosteleriá tradicional esto se resuelve separando los elementos de una comanda con una o varias líneas horizontales, que indicarían qué productos tienen que salir primero y cuales después, lo que se conoce formalmente como pasos de mesa.

Posteriormente, y ante la posibilidad de incluir **pagos virtuales**, comentó que al menos no lo implementaría para el cobro por parte del negocio, simplemente por la **desconfianza** que puede generar en los clientes que otra persona introduzca todos los datos de su tarjeta, incluido el CVV. Por parte de los clientes podría ser algo opcional, pero sería complicado controlar si una persona se va sin pagar pese a recibir una notificación de pago realizado en la propia aplicación.

Por último, surgió la pregunta de ¿qué ocurriría si se pierde la conexión a internet?, y me instó a tener en cuenta ese factor y tratar de proporcionar algún aviso y una alternativa manual.

Conclusiones

La valoración global de la sesión fue positiva, y sirvió para reafirmar la viabilidad del proyecto y su utilidad práctica.

Una vez analizadas las opiniones se decidió descartar la opción inicial del pago virtual, e incluir un mecanismo en la creación de comandas para tener en cuenta los pasos de mesa. Además, durante la elección de alternativas de herramientas y librerías para el desarrollo, se tendrá en cuenta el hecho de la posibilidad de pérdida de conexión en momentos puntuales.

Finalmente se quedó en volver a encontrase una vez estuviese el proyecto finalizado para realizar pruebas sobre el software y emitir una nueva valoración.

10.2.2 Focus group 2

Fecha: 23 de Junio de 2016

Persona Colaboradora: Jose Manuel García Fernández (extrabajador del sector, con experiencia durante más de 30 años en negocios de hostelería, por cuenta propia y ajena, desempeñando cargos de índole diversa).

Objetivos de la sesión

Tras una primera sesión aproximadamente un año antes, y después de haber quedado en volver a encontrase una vez que la aplicacación estuviese prácticamente terminada se establecen los siguientes objetivos para la nueva sesión:

- **Recordar la idea y el propósito de la aplicación**: Después de tanto tiempo es posible que Jose Manuel haya olvidado algunas características del proyecto. Por ello, se realizará una nueva explicación partiendo como base del resumen de la sesión anterior incidiendo en los cambios que se han realizado gracias a sus opiniones.

- **Realizar una serie de pruebas guiadas**: Se elaborará un documento con actividades guiadas, para que pueda probar los elementos más importantes de la aplicación y conocer en profundidad cómo funciona.

- **Obtener opiniones sobre cada una de las pruebas y posibles mejoras**: Para cada una de las actividades guiadas anteriores, se dejará un hueco para anotar las opiniones recibidas, y posibles errores o mejoras.

- **Valoración global y orientación comercial**: Una vez finalizadas las pruebas, se pedirá una valoración global a través de un formulario, y se preguntará cómo enfocaría la labor comercial en caso de querer vender el producto.

Actividades guiadas

A continuación, se muestra el formulario empleado para las actividades guiadas junto con sus resultados:

Actividad	Fácil	Regular	Difícil	Observaciones
Autentificarse	X			
Creación de una comanda: cabecera.	X			Cuando se juntan varias mesas, quizás separar los números de mesa por comas no es la mejor opcion
Creación de una comanda: selección de productos.	X			
Creación de una comanda: pasos de mesa y unidades.		X		Resulta muy útil la solución aportada para marcar pasos de mesa, pero sería conveniente indicar a través de un tutorial o similar que simbolizan sus iconos.
Vista de bar: marcar bebida preparada.	X			
Vista de cocina: marcar comida preparada.	X			
Prioridades negativas.	X			
Cobro de comanda.		X		La propina en un 99% de los casos no debería registrarse en la pantalla de facturación.
Visualización de estadísticas.	X			
Visualización CRM.	X			

Valoración general

Valoración de la usabilidad				
Facilidad de uso	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Sabe dónde se encuentra dentro de la aplicación?		X		
¿El sistema se comporta fiel a sus indicaciones?	X			
¿Le resulta sencillo el uso de la aplicación?	X			
Funcionalidad	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Funciona cada tarea como usted esperaba?		X		
¿El tiempo de respuesta es largo?				X
Valoración de la interfaz				
Aspectos gráficos	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Cree que los componentes tienen un tamaño adecuado?			X	
¿Los colores, imágenes e iconos son agradables?		X		
Diseño de la interfaz	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿La aplicación está bien estructurada?	X			
¿Cree que los componentes elegidos son adecuados?	X			
Valoración de la funcionalidad				
Características	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Cree que el modelo de comanda que utiliza la aplicación representa fielmente su utilidad real?	X			
¿Le resulta suficiente el sistema de cobros actual?			X	
¿Cree que las tablas de estadísticas iniciales son útiles?		X		
¿Considera conveniente la forma de ordenación de elementos por prioridad y su visión general en las vistas de barra y cocina?	X			
Valoración global	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Cree que la aplicación cuenta con suficientes características como para ofrecer valor en un negocio real?	X			
¿Cree que la aplicación puede ser más atractiva que otras soluciones actuales?		X		
Observaciones				
Es sistema de cobros necesita la generación de una factura susceptible de imprimir y entregar al cliente. Las estadísticas son de gran utilidad pero hay otros tipos que probablemente aportasen mayor valor al negocio.				

Conclusiones

Cabe destacar que Jose Manuel se acordaba perfectamente de la primera sesión, por lo que fue sencillo volver a recordar la idea para proceder a enseñar la aplicación y su funcionamiento.

La impresión general fue muy buena y realmente sorprendió pues se esperaba que fuese más complicada de utilizar, y un diseño muy diferente. En cuanto a la funcionalidad, llamaron la atención sobretodo las estadísticas, y la forma de anotar la comanda, con las unidades de los productos y la separación temporal entre ellos.

Como principales puntos a mejorar, se indicaron los siguientes:

- **Vista de sala**: Mostrar el número total de comensales. La separación de las mesas por comas puede confundir.

- **Cobros**: No incluir propina, ya que no forma parte de la facturación y no se registra, y proporcionar una factura imprimible para no tener que introducir los datos en caja de nuevo.

- **Estadísticas**: Incluir sobretodo la posibilidad de filtros de fechas en las estadísiticas de facturación, de forma que se pueda consultar la facturación de un día concreto o rango de fechas.

En cuanto a la orientación comercial, Jose Manuel sugiere comenzar con un piloto en **restaurantes de cierta envergadura**, pues es donde vé realmente el verdadero potencial del producto.

Una vez finalizadas las pruebas piloto, con buenas impresiones y un funcionamiento estable, probablemente lo expandiese a **hoteles medianos y grandes**, junto a más restaurantes. En este punto sugiere que sería necesario al menos una **persona de soporte** las 24h, ya que podrían surgir problemas, y los negocios deberían disponer de asistencia.

Como paso final, lo expandiría a **franquicias**, incluyendo **internacionalización** y pensando en el turismo extranjero, que sería un buen foco de ingresos.

10.3 Fragmentos de código fuente

En este apartado se incluye el código de las clases más importantes que componen la aplicación. Se ha realizado una selección con el fin de no saturar este documento, tratando de aportar ejemplos para ambos subsistemas.

10.3.1 Código Fuente del subsistema aplicación web para el negocio

10.3.1.1 Ejemplo de vistas

Product.html

```
<ion-toolbar dark>
    <ion-title>
        <span *ngIf="editing">Edit </span>
        <span *ngIf="!editing">New </span>
        <span *ngIf="isSection">Section</span>
        <span *ngIf="!isSection">Product</span>
    </ion-title>
</ion-toolbar>
<ion-content>
    <form>
        <ion-list>
            <ion-item>
                <ion-label>Type</ion-label>
                <ion-select [(ngModel)]="isSection">
                    <ion-option [value]="true">Section</ion-option>
                    <ion-option [value]="false">Product</ion-option>
                </ion-select>
            </ion-item>
            <ion-item *ngIf="isSection">
                <ion-label>Name</ion-label>
                <ion-input type="text" [(ngModel)]="section.name"></ion-input>
            </ion-item>
            <ion-item *ngIf="!isSection">
                <ion-label>Name</ion-label>
                <ion-input type="text" [(ngModel)]="product.name"></ion-input>
            </ion-item>
            <ion-item *ngIf="!isSection">
                <ion-label>Description</ion-label>
                <ion-textarea type="text" [(ngModel)]="product.description"></ion-
textarea>
            </ion-item>
            <ion-item *ngIf="!isSection">
                <ion-label>Price (€)</ion-label>
                <ion-input type="text" [(ngModel)]="product.price"></ion-input>
            </ion-item>
            <ion-item *ngIf="!isSection">
                <ion-label>Drink</ion-label>
                <ion-toggle [(ngModel)]="product.drink"></ion-toggle>
            </ion-item>
            <ion-item *ngIf="!isSection">
                <ion-label>Available</ion-label>
                <ion-toggle [(ngModel)]="product.available"></ion-toggle>
            </ion-item>
            <div padding>
                <button secondary block type="submit" (click)="addProduct()"
*ngIf="!editing">Add</button>
                <button secondary block type="submit" (click)="editProduct()"
*ngIf="editing">Edit</button>
                <button secondary block (click)="cancelProduct($event)">Cancel</button>
            </div>
        </ion-list>
    </form>
</ion-content>
```

Billing.html

```
<ion-navbar *navbar dark>
    <ion-title>
        Billing
    </ion-title>
</ion-navbar>

<ion-content>
    <form>
        <ion-card>
            <ion-card-header>
                Total Due
            </ion-card-header>
            <ion-card-content>
                <h1>{{ utils.getFormattedAmount(bill.amount) }}</h1>
            </ion-card-content>
        </ion-card>
        <ion-list radio-group [(ngModel)]="bill.method">
            <ion-list-header>
                PAYMENT METHOD
            </ion-list-header>

            <ion-item>
                <ion-icon name="cash" item-left></ion-icon>
                <ion-label>Cash</ion-label>
                <ion-radio value="cash" checked="true"></ion-radio>
            </ion-item>

            <ion-item>
                <ion-icon name="card" item-left></ion-icon>
                <ion-label>Credit Card</ion-label>
                <ion-radio value="card"></ion-radio>
            </ion-item>

            </ion-list>
             <ion-list>

            <ion-list-header>
                OTHER
            </ion-list-header>
            <ion-item>
                <ion-label>Tip</ion-label>
                <ion-input type="text" [(ngModel)]="bill.tip"></ion-input>
            </ion-item>
            <ion-item>
                <ion-label>Bill Ref.</ion-label>
                <ion-input type="text" [(ngModel)]="bill.reference"></ion-input>
            </ion-item>

        </ion-list>

        <div padding>
            <button secondary block type="submit" (click)="closeOrder()">Close
order</button>
        </div>
    </form>
</ion-content>
```

10.3.1.2 Ejemplo de controladores

Bar.ts

```
@Component({
    templateUrl: 'build/pages/bar/bar.html',
    providers: [OrderService, AuthService, UtilService]
})
export class BarPage {

    itemsToServe: {order: any, item: any}[];
    itemsReady: {order: any, item: any}[];

    constructor(
        public nav:NavController,
        private authService: AuthService,
        private orderService: OrderService,
        private util: UtilService
    ) {}

    ngOnInit() {
        this.authService.checkAuth().subscribe((user) => {
                if (user) {
                    let controller = this;
                    this.orderService.getItems(true, false).subscribe(items => {
                        setTimeout(function() {
                            controller.itemsToServe = items;
                        },500);
                    });
                    this.orderService.getItems(true, true).subscribe(items => {
                        setTimeout(function() {
                            controller.itemsReady = items;
                        },500);
                    });
                } else {
                    this.nav.setRoot(LoginPage);
                }
            }
        )
    }

    setReady(itemSnapshot: any, orderKey: string) {
        this.orderService.setItemAsReady(orderKey, itemSnapshot.key());
    }

}
```

Stats.ts

```
@Component({
    templateUrl: 'build/pages/admin/stats/stats.html',
    directives: [CHART_DIRECTIVES, NgClass, CORE_DIRECTIVES, FORM_DIRECTIVES],
    providers: [StatsService, AuthService, UtilService]
})
export class StatsPage {

    dailyRevenue:number = 0;
    monthlyRevenue:number = 0;

    peopleChartLabels:string[];
    peopleChartData:[{data: number[], label:string}];

    productChartLabels:string[];
    productChartData:[{data: number[], label:string}];

    hoursChartLabels:string[];
    hoursChartData:[{data: number[], label:string}];

    public chartOptions:any = {
        scaleShowVerticalLines: false,
        responsive: true
    };

    constructor(
```

```
        public nav: NavController,
        private statsService: StatsService,
        private authService: AuthService,
        private util: UtilService
    ) {}

    ngOnInit() {
        this.authService.checkAuth().subscribe((user) => {
            if (user && user.admin) {

                this.statsService.getPeopleStats().subscribe(value => {
                    this.peopleChartLabels = value.keys;
                    this.peopleChartData = [({data: value.values, label: "Daily
customers"})];
                });

                this.statsService.getProductStats().subscribe(value => {
                    this.productChartLabels = value.keys;
                    this.productChartData = [({data: value.values, label: "Times
ordered"})];
                });

                this.statsService.getLastMonthPopularHoursStats().subscribe(value => {
                    this.hoursChartLabels = value.keys;
                    this.hoursChartData = [({data: value.values, label: "Orders by
hour"})];
                });

                this.statsService.getDailyRevenue().subscribe(value => {
                    this.dailyRevenue = value;
                });

                this.statsService.getMonthlyRevenue().subscribe(value => {
                    this.monthlyRevenue = value;
                });

            } else {
                this.nav.setRoot(LoginPage);
            }
        });
    }

}
```

10.3.1.3 Ejemplo de clases modelo

User.ts

```
export class User {

    name:string;
    email:string;
    available:boolean;
    admin:boolean;
    uid:string;

    constructor() {
        this.name = "";
        this.email = "";
        this.available = true;
        this.admin = false;
    }

}
```

Product.ts

```
export class Product {

    name:string;
    description:string;
    price:number;
    drink:boolean;
    available:boolean;
    type:string;
    timesOrdered:number;

    constructor() {
        this.name = "";
        this.description = "";
        this.price = 0.00;
        this.drink = false;
        this.available = true;
        this.type = "product";
        this.timesOrdered = 0;
    }

}
```

10.3.2 Código fuente del subsistema aplicación móvil para el cliente

10.3.2.1 Ejemplo de servicios

FirebaseService.swift

```
import Foundation
import Firebase

class FirebaseService {

    static let sharedInstance = FirebaseService()

    private var ref: FIRDatabaseReference!

    private init() {
        self.ref = FIRDatabase.database().reference()
    }

    func getRestaurant(restaurantId : String, completion: (restaurant: Restaurant) ->
Void) {

ref.child("businesses").child(restaurantId).observeSingleEventOfType(FIRDataEventType.Va
lue, withBlock: { (snapshot) in
            let json = snapshot.value as! [String : AnyObject]
            let restaurant = self.parseRestaurant(restaurantId, json: json)
            completion(restaurant: restaurant)
        })
    }

    func getMenu(completion: (menu: Product) -> Void) {
        ref.child("products").observeSingleEventOfType(FIRDataEventType.Value,
withBlock: { (snapshot) in
            let json = snapshot.value as! [String : AnyObject]
            let root = Product(id: "root", name: "Menu", type: ProductType.Section)
            for jsonProducts in json {
                root.children.append(self.parseProduct(jsonProducts.0, json:
jsonProducts.1 as! Dictionary<String, AnyObject>, parent: root))
            }
            completion(menu: root)
        })
    }
```

```
    func subscribeOrder(orderKey: String, completion: (order: Order) -> Void) {
        ref.child("orders").child(orderKey).observeEventType(FIRDataEventType.Value,
withBlock: {(snapshot) in
            let json = snapshot.value as! [String : AnyObject]
            let order = self.parseOrder(json)
            completion(order: order)
        })
    }

    func placeOrder(order: Order, completion: (key: String) -> Void) {
        let key = ref.child("orders").childByAutoId().key
        let jsonOrder = generateOrderDict(order)
        ref.child("orders").child(key).setValue(jsonOrder, withCompletionBlock:
{(NSError, FIRDatabaseReference) -> Void in
            completion(key: key)
        })
    }

    private func parseRestaurant(key: String, json: Dictionary<String,AnyObject>) ->
Restaurant {
        let id = key
        let name = json["name"] as! String
        let description = json["description"] as! String
        return Restaurant(id: id, name: name, description: description)
    }

    private func parseProduct(key: String, json: Dictionary<String,AnyObject>, parent:
Product?) -> Product {
        let id = parent == nil || parent?.id == "root" ? "/\(key)" :
"\(parent!.id)/children/\(key)"
        let name = json["name"] as! String
        let type = json["type"] as! String == "section" ? ProductType.Section :
ProductType.Product
        let product = Product(id: id, name: name, type: type)
        product.parent = parent
        switch type {
        case ProductType.Product:
            let description = json["description"] as! String
            let price = (json["price"] as! NSString).floatValue
            product.description = description
            product.price = price
        case ProductType.Section:
            for child in json["children"] as! [String : AnyObject] {
                product.children.append(parseProduct(child.0, json: child.1 as!
Dictionary<String, AnyObject>, parent: product))
            }
        }
        return product
    }

    private func parseOrder(json: Dictionary<String,AnyObject>) -> Order {
        let table = (json["tables"] as! NSString).integerValue
        let order = Order(table: table)
        order.open = json["open"] as! Bool
        order.pax = json["pax"] as! Int
        order.status = json["status"] as! String
        order.time = Int64(json["time"] as! Int)
        for jsonItem in json["items"] as! [AnyObject] {
            let key = jsonItem["key"] as! String
            let jsonMeta = jsonItem["meta"] as! [String : AnyObject]
            let name = jsonMeta["name"] as! String
            let units = jsonMeta["units"] as! Int
            let time = jsonMeta["time"] as! Int
            let served = jsonMeta["served"] as! Bool
            let ready = jsonMeta["ready"] as! Bool
            let item = Item(key: key, name: name, units: units, time: time)
            item.served = served
            item.ready = ready
            order.items.append(item)
        }
        return order
    }

    private func generateOrderDict(order: Order) -> [String : AnyObject] {
        var jsonOrder: [String: AnyObject] = [:]
        jsonOrder["open"] = order.open
        jsonOrder["pax"] = order.pax
        jsonOrder["status"] = order.status
```

```
        jsonOrder["tables"] = String(order.table)
        jsonOrder["time"] = Int(order.time)
        var jsonItems : [String: AnyObject] = [:]
        for i in 0..<order.items.count {
            let item = order.items[i]
            var jsonItem: [String: AnyObject] = [:]
            jsonItem["key"] = item.key
            var jsonMeta: [String: AnyObject] = [:]
            jsonMeta["name"] = item.name
            jsonMeta["ready"] = item.ready
            jsonMeta["served"] = item.served
            jsonMeta["time"] = item.time
            jsonMeta["units"] = item.units
            jsonItem["meta"] = jsonMeta
            jsonItems[String(i)] = jsonItem
        }
        jsonOrder["items"] = jsonItems
        return jsonOrder
    }

}
```

10.3.2.2 Ejemplo de controladores

CartController.swift

```
import Foundation
import UIKit

class CartController : UIViewController, ItemSelectionDelegate {

    let fs = FirebaseService.sharedInstance

    @IBOutlet weak var restaurantTitle: UILabel!
    @IBOutlet weak var browseButton: UIButton!
    @IBOutlet weak var placeOrderButton: UIButton!
    @IBOutlet weak var tableView: UITableView!
    @IBOutlet weak var emptyMessage: UILabel!

    var order: Order?
    var restaurant : Restaurant?
    var table : Int?
    var orderKey : String?

    override func viewDidLoad() {
        super.viewDidLoad()
        placeOrderButton.layer.cornerRadius = 3;
        browseButton.layer.cornerRadius = 3;
        browseButton.layer.borderWidth = 1;
        browseButton.layer.borderColor = UIColor.colorFromHex("#3DC2AC").CGColor
        tableView.delegate = self
        tableView.dataSource = self
        restaurantTitle.text = restaurant!.name
        order = Order(table: table!)
    }


    @IBAction func browseMenu(sender: UIButton) {
        self.performSegueWithIdentifier("showMenu", sender: self)
    }

    @IBAction func placeOrder(sender: UIButton) {
        let alertController = UIAlertController(title: "Place order", message: "You're
about to create a new order, please, make sure you've included everything, fill in how
many people you are and place the order.", preferredStyle: UIAlertControllerStyle.Alert)
        let saveAction = UIAlertAction(title: "Place order", style:
UIAlertActionStyle.Default, handler: {
            alert -> Void in
            let textField = alertController.textFields![0] as UITextField
            let pax = Int(textField.text!)
            if pax != nil {
                self.order!.pax = pax!
            }
```

```
            self.order!.time = Int64(NSDate().timeIntervalSince1970 * 1000)
            self.fs.placeOrder(self.order!, completion: { (key: String) -> Void in
                self.orderCompleted(key)
            })
        })
        let cancelAction = UIAlertAction(title: "Cancel", style:
UIAlertActionStyle.Default, handler: nil)
        alertController.addTextFieldWithConfigurationHandler { (textField :
UITextField!) -> Void in
            textField.placeholder = "Number of people"
            textField.keyboardType = UIKeyboardType.NumberPad
        }
        alertController.addAction(saveAction)
        alertController.addAction(cancelAction)
        alertController.preferredAction = saveAction
        self.presentViewController(alertController, animated: true, completion: nil)
    }

    func orderCompleted(key: String) {
        orderKey = key
        self.performSegueWithIdentifier("showStatus", sender: self)
    }

    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
        if(segue.identifier == "showMenu") {
            let menuController = (segue.destinationViewController as! MenuController)
            menuController.itemSelectionDelegate = self
        } else if(segue.identifier == "showStatus") {
            let statusController = (segue.destinationViewController as!
StatusController)
            statusController.orderKey = orderKey
        }
    }

    func onItemAdded(item: Item) {
        order!.items.append(item)
        tableView.reloadData()
    }

}

extension CartController: UITableViewDataSource {

    func numberOfSectionsInTableView(tableView: UITableView) -> Int {
        return 1
    }

    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        let rows = self.order!.items.count
        emptyMessage.hidden = rows > 0
        placeOrderButton.hidden = rows == 0
        return rows
    }

    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath)
-> UITableViewCell {
        let cell = UITableViewCell()
        cell.selectionStyle = UITableViewCellSelectionStyle.None
        cell.accessoryType = UITableViewCellAccessoryType.DisclosureIndicator
        let item = self.order!.items[indexPath.row]
        let time = item.time == 0 ? "ASAP" : "\(item.time) min"
        cell.textLabel?.text = "\(item.units) x \(item.name) (\(time))"
        return cell
    }

}

extension CartController: UITableViewDelegate {

    func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath:
NSIndexPath) {
    }

    func tableView(tableView: UITableView, commitEditingStyle editingStyle:
UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath) {
        if editingStyle == .Delete {
            tableView.beginUpdates()
            order!.items.removeAtIndex(indexPath.row)
            tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Automatic)
```

```
            tableView.endUpdates()
        }
    }

}

extension UIColor {
    static func colorFromHex(hexString: String, alpha: CGFloat = 1) -> UIColor {
        if hexString.characters.count < 7 {
            return UIColor.whiteColor()
        }
        let hexStringWithoutHash =
hexString.substringFromIndex(hexString.startIndex.advancedBy(1))

        let eachColor = [

hexStringWithoutHash.substringWithRange(hexStringWithoutHash.startIndex...hexStringWitho
utHash.startIndex.advancedBy(1)),

hexStringWithoutHash.substringWithRange(hexStringWithoutHash.startIndex.advancedBy(2)...
hexStringWithoutHash.startIndex.advancedBy(3)),

hexStringWithoutHash.substringWithRange(hexStringWithoutHash.startIndex.advancedBy(4)...
hexStringWithoutHash.startIndex.advancedBy(5))]

        let hexForEach = eachColor.map {CGFloat(Int($0, radix: 16) ?? 0)}
        return UIColor(red: hexForEach[0] / 255, green: hexForEach[1] / 255, blue:
hexForEach[2] / 255, alpha: alpha)
    }
}
```

ScanController.swift

```
class ScanController: UIViewController, QRCodeReaderViewControllerDelegate {

    lazy var readerVC = QRCodeReaderViewController(metadataObjectTypes:
[AVMetadataObjectTypeQRCode])

    let fs = FirebaseService.sharedInstance

    @IBOutlet weak var backgroundScanImage: UIImageView!

    var backgroundImageTimer: NSTimer!
    var backgroundIndex = 1

    var restaurant : Restaurant?
    var table : Int?

    override func viewDidLoad() {
        super.viewDidLoad()
        backgroundImageTimer = NSTimer.scheduledTimerWithTimeInterval(0.8, target: self,
selector: #selector(ScanController.changeBackground), userInfo: nil, repeats: true)
        let backgroundTap = UITapGestureRecognizer(target: self, action:
#selector(ScanController.launchScan))
        backgroundTap.numberOfTapsRequired = 1
        backgroundScanImage.userInteractionEnabled = true
        backgroundScanImage.addGestureRecognizer(backgroundTap)
    }

    func launchScan() {
        readerVC.delegate = self
        readerVC.modalPresentationStyle = .FormSheet
        presentViewController(readerVC, animated: true, completion: nil)
    }

    func changeBackground() {
        if (backgroundIndex == 1) {
            backgroundScanImage.image = UIImage(named: "savour_scan_2.png")
            backgroundIndex = 2
        } else {
            backgroundScanImage.image = UIImage(named: "savour_scan_1.png")
            backgroundIndex = 1
        }
    }

    func showCart() {
        self.dismissViewControllerAnimated(true, completion: {
```

```
            self.performSegueWithIdentifier("showCart", sender: self)
        })
    }

    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
        if(segue.identifier == "showCart") {
            let cartController = (segue.destinationViewController as! CartController)
            cartController.restaurant = restaurant
            cartController.table = table
        }
    }

    // QRCodeReader Delegate Methods

    func reader(reader: QRCodeReaderViewController, didScanResult result:
QRCodeReaderResult) {
        let resultArray = result.value.characters.split(";")
        let restaurantId = String(resultArray.first!)
        self.table = (String(resultArray.last!) as NSString).integerValue
        fs.getRestaurant(restaurantId, completion: {(restaurant: Restaurant) -> Void in
            self.restaurant = restaurant
            self.showCart()
        })
    }

    func readerDidCancel(reader: QRCodeReaderViewController) {
        self.dismissViewControllerAnimated(true, completion: nil)
    }

}
```

10.3.2.3 Ejemplo de modelo

Product.swift

```
import Foundation

class Product {

    var id: String
    var name: String
    var description: String?
    var price: Float?
    var type: ProductType
    var parent: Product?
    var children: [Product]

    init(id: String, name: String, type: ProductType) {
        self.id = id
        self.name = name
        self.type = type
        self.children = []
    }

}

enum ProductType {
    case Product
    case Section
}
```

Item.swift

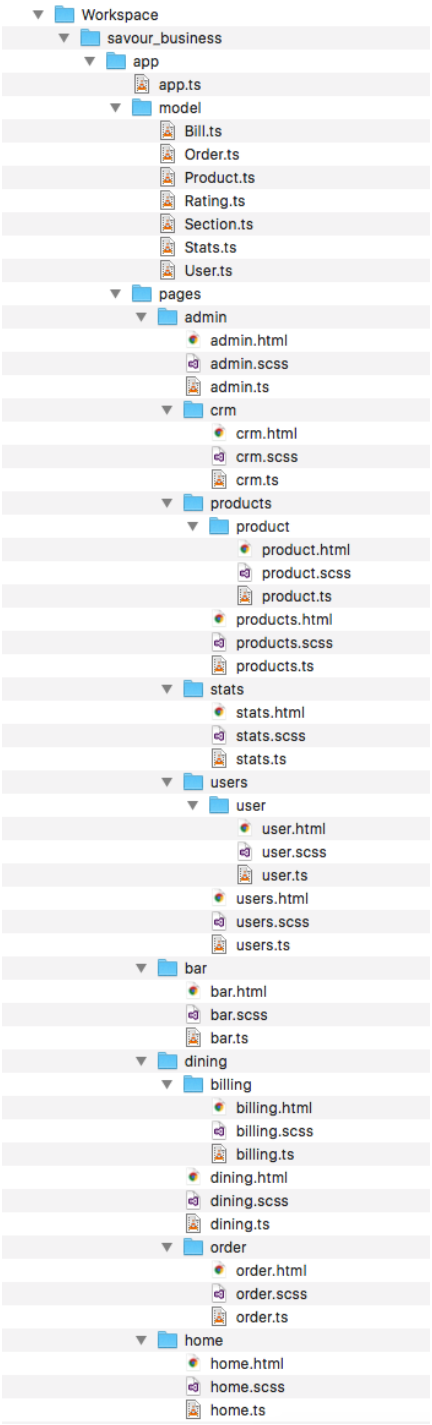
```
import Foundation

class Item {

    var key: String
    var name: String
    var units: Int
    var time: Int
    var served: Bool
    var ready: Bool

    init(key: String, name: String, units: Int, time: Int) {
        self.key = key
        self.name = name
        self.units = units
        self.time = time
        self.served = false
        self.ready = false
    }

}
```

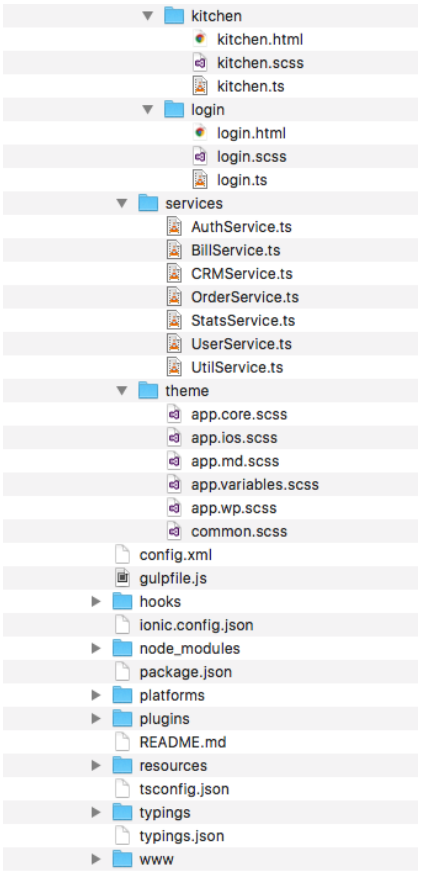
10.4 Contenido del anexo de código fuente

Junto con este proyecto, se entrega un fichero comprimido con el código fuente de la aplicación. Dicho fichero contiene los proyectos para los subsistemas aplicación para el negocio y aplicación para los clientes, desarrollados con IONIC y XCode respectivamente.

A continuación, se puede ver la estructura general de ambos proyectos.

Proyecto aplicación web para el negocio





Para realizar la **instalación y despliegue** basta con descargar NodeJS 5, IONIC Framework 2, situarse sobre la raíz del proyecto y ejecutar los siguientes comandos sobre una terminal:

```
npm install
ionic serve --lab
```

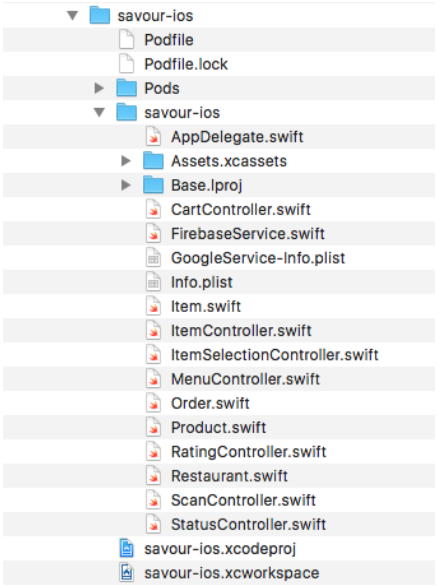
Cuando finalice el proceso, se lanzará el navegador por defecto, escuchando en el puerto 8100.

Para acceder a la aplicación con un usuario administrador, pueden utilizarse las siguientes credenciales:

- Email: miw@savour.com
- Password: miw2016

Proyecto aplicación móvil para los clientes

Cabe destacar que, aunque parezca una estructura plana de ficheros, XCode internamente los agrupa según se hayan definido durante el desarrollo.



Para ejecutar la aplicación es necesario instalar XCode, abrir el fichero de proyecto **savour-ios.xcworkspace**, seleccionar un dispositivo físico o emulador y ejecutar una nueva sesión de debug o release.