UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE
TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

TÉCNICAS PARA DESARROLLO DE APLICACIONES MULTIDISPOSITIVO PARA EMPRESAS E IMPLEMENTACIÓN DE CASO DE USO PARA UN RESTAURANTE

Titulación: Ingeniero de Telecomunicación
Autor: Carlos Pérez Pérez
Tutor: Luis Hernández Acosta
Fecha: Junio de 2015

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE
TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

TÉCNICAS PARA DESARROLLO DE APLICACIONES MULTIDISPOSITIVO PARA EMPRESAS E IMPLEMENTACIÓN DE CASO DE USO PARA UN RESTAURANTE

HOJA DE FIRMAS

Alumno

Fdo.: Carlos Pérez Pérez

Tutor

Fdo.: Luis Hernández Acosta

Fecha: Junio de 2015

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN
Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

TÉCNICAS PARA DESARROLLO DE APLICACIONES MULTIDISPOSITIVO PARA EMPRESAS E IMPLEMENTACIÓN DE CASO DE USO PARA UN RESTAURANTE

HOJA DE EVALUACIÓN

Calificación:

Presidente

Fdo.:

Vocal **Secretario**

Fdo.: Fdo.:

Fecha: Junio de 2015

Agradecimientos

No se ni cuántas veces habré empezado esta sección del proyecto en mi cabeza, e incluso llegué a tenerla bastante clara, pero luego pensé que no estaría bien que la parte de agradecimientos del proyecto fuese (afortunadamente) más larga que el propio proyecto, así que decidí hacerlo cortito, aún sabiendo que me dejaría mucha gente atrás.

En primer lugar, cómo no, darle las gracias a mis padres por estar siempre detrás de mí incondicionalmente. Nunca lo podré agradecer suficiente.

A Marta, compañera de todo durante estos increíbles años y posiblemente la mejor persona que conozco. No te alejes nunca de mí.

Por último a mis compañeros de carrera, mi segunda familia. Ustedes saben que no hubiese llegado hasta aquí sin ustedes. Que nuestros caminos no se separen demasiado

A todos, muchas gracias.

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1. Introducción

En este capítulo se presentan los antecedentes y el estado del arte que ha dado lugar al planteamiento de este Proyecto Fin de Carrera, así como los objetivos principales que se pretenden satisfacer con su desarrollo.

Además, se presenta la estructura del contenido del presente documento con el fin de proporcionar una idea global del trabajo realizado.

1.1 Antecedentes

En los últimos años, el mercado de los dispositivos móviles ha sufrido un gran crecimiento, con gran cantidad de cambios respecto a como solía ser. Sin ir más lejos y por poner un ejemplo, la cuota de mercado de Nokia en 2006 era de un 50% de los dispositivos vendidos en el mundo, con solo un 14% de Samsung. En la actualidad estos datos han cambiado de forma que Samsung se erige como el gran dominador del mercado con un 31% de cuota frente al 15% de Nokia, aunque de ese 15%, solo un 3% se debe a la venta de los llamados smartphones, teléfonos cuyas principales características son el acceso a Internet, el uso de pantallas táctiles y la posibilidad de instalar las aplicaciones que el usuario desee.[1][2].

A este respecto, debemos hablar del principal caballo de batalla entre los distintos fabricantes a nivel mundial, como son los sistemas operativos. En la actualidad, el sistema operativo Android, sistema de software libre basado en Linux, perteneciente a Google que se ha convertido en el sistema operativo de referencia, obteniendo en el primer cuatrimestre de 2013, una cuota de mercado del 79% de los teléfonos inteligentes.[3]

En esta lista, le siguen iOS, sistema operativo de Apple, presente en los teléfonos y tablets de la empresa, que copa el 14% del mercado, y otras como Windows Phone y BlackBerry, con un 3,7 y 2,9%, respectivamente.

Como vemos, a pesar de que Android actualmente es el gran dominador del mercado, existen varios sistemas operativos con una cuota de mercado importante, ya que apenas un 3,7% suponen 8,7 millones de dispositivos vendidos en el último año.

Top Smartphone Operating Systems, Shipments, and Market Share, 2013 Q3 (Units in Millions)

Operating System	2Q13 Unit Shipments	2Q13 Market Share	2Q12 Unit Shipments	2Q12 Market Share	Year-over-Year Change
Android	187.4	79.3%	108	69.1%	73.5%
iOS	31.2	13.2%	26	16.6%	20.0%
Windows Phone	8.7	3.7%	4.9	3.1%	77.6%
BlackBerry OS	6.8	2.9%	7.7	4.9%	-11.7%
Linux	1.8	0.8%	2.8	1.8%	-35.7%
Symbian	0.5	0.2%	6.5	4.2%	-92.3%
Others	N/A	0.0%	0.3	0.2%	-100.0%
Total	**236.4**	**100.0%**	**156.2**	**100.0%**	**51.3%**

Source: IDC Worldwide Mobile Phone Tracker, August 7, 2013

Figura 1-1-1. Cuota de mercado de sistemas operativos (Agosto ‘13)

Como podría parecer evidente, cada una de estos sistemas operativos, implican una forma de programación y diseño diferentes entre sí, cada una con sus propias características, lo que supondría grandes costes de desarrollo a empresas que desearan desplegar una misma aplicación para cada una de las plataformas, ya que implicaría la contratación de desarrolladores especializados en cada sistema.

Por otro lado, a la hora de diseñar aplicaciones, los desarrolladores se encuentran con otro problema, que no es otro que el distinto tamaño de las muchas pantallas que pueblan el mercado de los teléfonos inteligentes, abarcando desde las 3.5 pulgadas de los modelos más pequeños de teléfono, hasta las 10 pulgadas que encontramos en las tablets, pasando por los denominados “phablets”, dispositivos de tamaño intermedio con una dimensión en torno a las 6 pulgadas.

A esto hay que añadirle las distintas resoluciones disponibles en el mercado, desde los 400x840 pixeles en los primeros modelos, hasta los 1920x1080 pixeles en alta definición que ofrecen los modelos más avanzados.

Estas grandes diferencias de tamaño y resolución, como decíamos, suponen un dolor de cabeza para los desarrolladores, ya que implican prácticamente hacer un diseño específico para cada tipo de pantalla y resolución.

Figura 1-1-2. Comparación tamaños de pantalla[4]

Es por esto que se hace necesario el uso de herramientas de diseño de aplicaciones universales, es decir, que podamos diseñar una vez y que podamos usar en todos los dispositivos, sea cual sea su sistema operativo, tamaño y resolución de pantalla, ahorrando una gran cantidad de dinero a las empresas desarrolladoras y contribuyendo a una mejor experiencia de usuario, al estar estandarizadas de un dispositivo a otro.

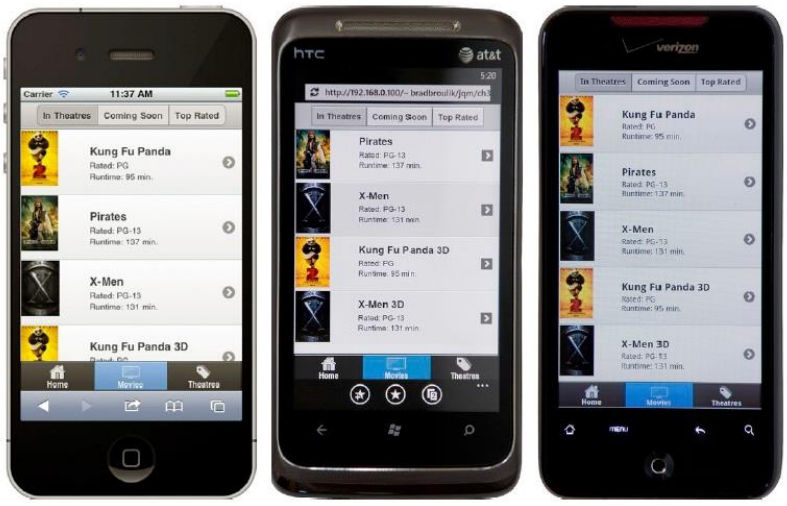

Figura 1-1-3. Aplicación desarrollada en 3 plataformas distintas de forma estandarizada

1.2 Objetivo

El objetivo principal de este proyecto es el diseño de una aplicación móvil siguiendo un flujo de desarrollo en el que, usando las herramientas adecuadas, conseguiremos que dicha aplicación pueda ser utilizada en cualquier tipo de dispositivo móvil, sea cual sea su sistema operativo, tamaño y resolución.

Dicha globalidad se conseguirá mediante la creación de una aplicación web, que será ejecutada en los dispositivos sin ningún problema, ya que todos los dispositivos del mercado contienen un navegador web, independientemente del fabricante, lo que hará que nuestra aplicación sea ejecutable en los mismos. Además de esto, al obtener una aplicación web, la misma puede ser ejecutada desde un PC accediendo al navegador clásico de cualquier ordenador.

La aplicación a desarrollar se trata de un sistema con el que conectar en tiempo real a los trabajadores de un restaurante, de forma que agilizaremos el intercambio de información entre ellos, con el objetivo de mejorar la eficiencia del servicio. En este sistema, el camarero deberá llevar un dispositivo móvil con la aplicación instalada, siendo este el encargado de tomar nota de las mesas al estilo tradicional, pero informatizado.

Para ello, la aplicación que el camarero llevará en la mano, deberá tener implementada la carta del restaurante (almacenada además en el servidor del restaurante), a fin de que, tomar notas se lleve a cabo simplemente pulsando sobre el elemento que los clientes hayan pedido, otorgando sencillez y rapidez a la app, y en consecuencia, al sistema de atención del restaurante. De esta forma, y teniendo en cuenta la posibilidad de que ciertos platos puedan agotarse a lo largo del día, la carta que el camarero lleve tendrá actualizado qué platos siguen disponibles y cuáles están agotados, evitando de esta manera una petición que pueda ser rechazada por la cocina y permitiendo mantener controlados los productos que sea necesario pedir en cada momento para continuar con el funcionamiento correcto del servicio de restaurante. Esto se conseguirá incluyendo un perfil de administrador que podrá actualizar en tiempo real el menú del día o suprimir los platos agotados.

Dichas notas serán enviadas bien a la cocina donde el cocinero recibirá en orden, en una pantalla táctil de su tablet o en un monitor de cualquier PC de sobremesa, los platos a cocinar según se tomen las notas, o bien a la barra del bar (posibilitándose el uso de los mismos dispositivos), donde otro empleado del restaurante preparará las bebidas que la mesa pida, para que sean llevadas a la misma. Además, el sistema llevará a cabo un sistema de actualizaciones con el que todos los empleados del restaurante se dan por enterados de la orden que les ha llegado desde el teléfono del camarero que realizo la nota.

La forma en la que se implementará esto se basará en un esquema cliente-servidor, donde los distintos dispositivos se conectarán por Internet o WiFi a la base de datos en la nube. De esta manera, lograremos mejorar la eficiencia del servicio, manteniendo la coordinación entre los empleados del restaurante sin que sea necesaria comunicación oral de ningún tipo, realizándose todas las notificaciones necesarias por vía telemática.

1.3 Organización de la Memoria

El presente documento está compuesto por cuatro partes claramente diferenciadas: memoria descriptiva, anexos, presupuesto y pliego de condiciones. La memoria descriptiva se divide en 8 capítulos, además de la bibliografía utilizada. La estructura de los capítulos y el contenido de cada uno de ellos se resumen a continuación:

- Capítulo 1. Introducción. Este capítulo recoge la introducción y los antecedentes que han dado lugar a la propuesta y realización de este Proyecto Fin de Carrera, así como los objetivos del mismo y la estructura del documento.

- Capítulo 2. Tecnologías software. En este capítulo se expondrán las tecnologías a utilizar, sus principales características por las que son necesarias para su inclusión en este proyecto, y las razones que las hacen preferibles a otras soluciones del mercado.

- Capítulo 3. Descripción general del funcionamiento de la aplicación desarrollada, apoyándose en diagramas de bloques para su mejor comprensión.

- Capítulo 4. Descripción detallada de los casos de uso del sistema implementado, donde basándonos en diagramas de bloques modulares, se explicará la interacción entre los usuarios con el objetivo de cumplir con el objetivo planteado.

- Capítulo 5. Base de datos. En este apartado se explicará la base de datos diseñada para el correcto funcionamiento del sistema, así como la forma en que se conecta el sistema a la base de datos remota.

- Capítulo 6. Implementación modular. Descripción detallada de cada uno de los roles que intervienen en el sistema, apoyándose en diagramas de flujo para una mejor comprensión y evaluación.

- Capítulo 7. Pruebas y resultados. Capítulo donde se expondrán las tareas llevadas a cabo para la comprobación del correcto funcionamiento del sistema desarrollado, así como los resultados de las mismas.

- Capítulo 8. Conclusiones y líneas futuras. En este apartado analizaremos los resultados de las pruebas realizadas. Además, comprobaremos si hemos obtenido los resultados deseados en relación a los objetivos del proyecto. Por último se expondrán algunas líneas de desarrollo futuro.

CAPÍTULO 2. Tecnologías software

En este capítulo se expondrán las tecnologías a utilizar, sus principales características por las que son necesarias para su inclusión en este proyecto, y las razones que las hacen preferibles a otras soluciones del mercado.

2.1 HTML5

HTML, siglas de *HyperText Markup Language* (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia para la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código *HTML*) para la definición de contenido de una página web, como texto o imágenes, entre otros. Es un estándar a cargo de la W3C, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación.[5]

El lenguaje *HTML* basa su filosofía de desarrollo en la referenciación. Para añadir un elemento externo a la página (imagen, vídeo, *script*, entre otros.), este no se incrusta directamente en el código de la página, sino que se hace una referencia a la ubicación de dicho elemento mediante texto. De este modo, la página web contiene sólo texto mientras que recae en el navegador web (interpretador del código) la tarea de unir todos los elementos y visualizar la página final. Al ser un estándar, *HTML* busca ser un lenguaje que permita que cualquier página web escrita en una determinada versión, pueda ser interpretada de la misma forma (estándar) por cualquier navegador web actualizado.

Sin embargo, a lo largo de sus diferentes versiones, se han incorporado y suprimido diversas características, con el fin de hacerlo más eficiente y facilitar el desarrollo de páginas web compatibles con distintos navegadores y plataformas (PCs, portátiles, o el caso que nos ocupa en este proyecto, teléfonos inteligentes y tabletas).

HTML5 es la versión actualizada del estándar *HTML* que establece una serie de nuevos elementos y atributos orientados al uso actual de los sitios web modernos y la supresión de algunos elementos ya obsoletos de la versión 4. Un ejemplo de estas nuevas funcionalidades son las etiquetas <audio> y <video>, cuyo objetivo principal es facilitar la integración de elementos multimedia en los sitios web a través del uso de una interfaz estandarizada.[6]

De esta forma, *HTML* nos servirá como el "esqueleto" sobre el que implementar nuestras funcionalidades, dejando para *Javascript* y *jQuery*, la carga lógica de la página web, como veremos a continuación.

2.2 JavaScript

JavaScript es un lenguaje de programación interpretado y orientado a objetos. *JavaScript* se diseñó con una sintaxis similar al C, aunque adopta nombres y convenciones del lenguaje de programación *Java*. Sin embargo *Java* y *JavaScript* no están relacionados y tienen semánticas y propósitos diferentes.

Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web, permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

Todos los navegadores modernos interpretan el código *JavaScript* integrado en las páginas web. Para interactuar con una página web se provee al lenguaje *JavaScript* de una implementación del *Document Object Model* (DOM), esto es, *JavaScript* es capaz de leer las estructuras creadas en un fichero *HTML* y modificarlas, lo que nos da una enorme libertad para poder cambiar las interfaces de usuario.[7]

En este punto, es importante introducir el concepto de AJAX, acrónimo de *Asynchronous JavaScript And XML* (JavaScript asíncrono y XML), una técnica de desarrollo web para crear aplicaciones interactivas o RIA (*Rich Internet Applications*). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página.[8]

La razón por la que es importante tener noción de lo que AJAX ofrece y nos permite hacer es porque la API (*Application Programming Interface*) de Parse que se utilizará para conectarnos con nuestra base de datos remota, utiliza este método de conexión, cosa que se explicará más en profundidad en el apartado 2.5.

2.3 jQuery

jQuery es una librería escrita en *JavaScript* que se ha convertido en los últimos años prácticamente en un lenguaje de programación de referencia debido a la facilidad de uso y lo extendido de su uso a nivel mundial. *jQuery* es software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de *GNU* v2, permitiendo su uso en proyectos libres y privativos.

Su principal característica es la sencillez de uso, ya que reduce muchísimo el número de líneas de código que necesitaríamos si usáramos *JavaScript* nativo, especialmente en funcionalidades como realizar llamadas *AJAX*, manipulación del *DOM* del *HTML,* en aspectos como la inserción o supresión de elementos, o manipulación de eventos tales como un *click* en pantalla.[9][10]

2.4 CSS

Hoja de estilo en cascada o *CSS* (*cascading style sheets*) es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en *HTML* o *XML*. El *World Wide Web Consortium* (*W3C*) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

La idea que se encuentra detrás del desarrollo de *CSS* es separar la estructura de un documento de su presentación, es decir, separar las estructuras creadas en *HTML*, del estilo que tendrán dichas estructuras. Un ejemplo rápido sería un párrafo de texto plano en *HTML*, al que usando *CSS* podríamos especificar el tipo y tamaño de letra, el color o los márgenes a utilizar.

La información de estilo puede ser definida en un documento separado o en el mismo documento *HTML*. En este último caso podrían definirse estilos generales en la cabecera del documento o en cada etiqueta particular mediante el atributo «*style*» del *HTML*.[11]

Figura 2-4-1. Ejemplo de página web sin y con CSS activado

Por otro lado, la inclusión de CSS nos permitirá adaptar nuestro interfaz de usuario al dispositivo en el que se mostrará, siendo esto capital en nuestro sistema, ya que se pretende que sea utilizable en cualquier dispositivo, sea cual sea su tamaño de pantalla, como ya se explicó en la introducción del proyecto.

2.5 Parse

Parse es una base de datos no relacional, basada en MongoDB, una base de datos usada especialmente en desarrollo web por su velocidad. Parse, además, es una base de datos “en la nube”. Esto hará que no sea necesario que tengamos ningún tipo de servidor propio para almacenar nuestra información. Parse es un sistema que ofrece una suscripción gratuita para la primera aplicación desarrollada usando sus servidores siempre que no se sobrepase un número de peticiones por segundo que en nuestro caso de uso será prácticamente imposible alcanzar (30 peticiones por segundo).[12]

Además de esto, Parse ofrece una muy buena *API* con la que abstraer a nuestra aplicación de las funciones de red que realiza para conectarse con su base de datos remota. Por otro lado, dicha *API* cuenta con una muy buena documentación con la que podemos empezar a desarrollar nuestra aplicación prácticamente desde el primer momento.

2.6 PhoneGap

Al igual que con *jQuery*, *PhoneGap* es una librería para el desarrollo de aplicaciones móviles que permite a los programadores desarrollar aplicaciones para dispositivos móviles utilizando herramientas genéricas tales como *JavaScript*, *HTML* y *CSS*. Las aplicaciones resultantes son híbridas, es decir que no son realmente aplicaciones nativas al dispositivo (ya que el renderizado es realizado mediante vistas web y no con interfaces gráficas específicas a cada sistema), pero no se tratan tampoco de aplicaciones web (teniendo en cuenta que son aplicaciones que son empaquetadas para poder ser desplegadas en el dispositivo incluso trabajando con el *API* del sistema nativo).

En el libro, además de detallar las capacidades de *PhoneGap*, indicando qué partes de un teléfono podemos usar, podemos encontrar el siguiente gráfico, que nos ofrece una visión de cómo funciona *PhoneGap*, combinándose con *jQuery*

(biblioteca *JavaScript*), para obtener la aplicación en los diferentes dispositivos, con diferentes sistemas operativos.[13]

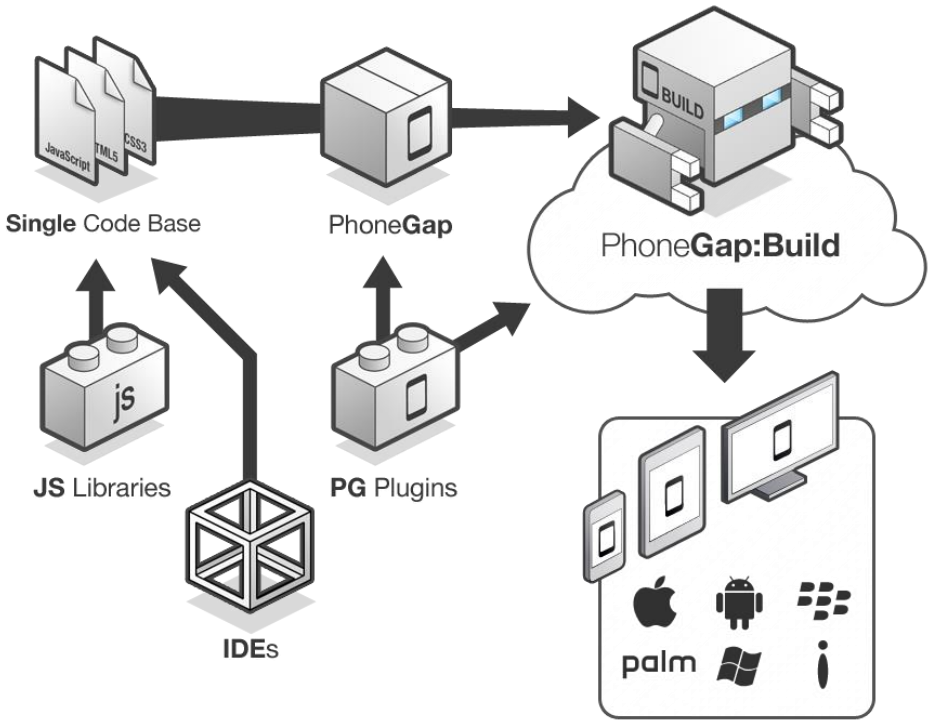

Figura 2-6-1. Esquema de funcionamiento de PhoneGap

PhoneGap maneja *APIs* que permiten tener acceso a elementos como el acelerómetro, cámara, contactos en el dispositivo, red, almacenamiento, notificaciones, etc. Por otro lado, *PhoneGap* además nos permite que el desarrollo sea ejecutando las aplicaciones en nuestro navegador web, sin tener que utilizar un simulador dedicado a esta tarea, lo que nos ahorra gran cantidad del tiempo de arranque que implican dichos emuladores debido a la alta carga computacional necesaria para ello.

La principal ventaja que esta herramienta nos proporciona es la generación de un único flujo de diseño e implementación para cubrir las múltiples plataformas del mercado, lo que lo hace una muy buena alternativa para las empresas dedicadas al

desarrollo de aplicaciones para dispositivos móviles, ya que pueden disminuir significativamente los costes, al no tener que cargar con los costes derivados del desarrollo de varias aplicaciones en distintos dispositivos, ya sea en forma de sueldos de desarrolladores, de diseñadores de interfaces, o de equipos.

CAPÍTULO 3. Descripción general

En este capítulo se mostrará, con el apoyo de diagramas de flujo y de bloques, la funcionalidad general del sistema a implementar.

Llegados a este punto, el lector habrá entendido que el propósito de este Proyecto Fin de Carrera es el diseño e implementación de una aplicación cliente-servidor para un restaurante, con la que los empleados del mismo puedan interactuar entre sí, y con la que podrán agilizar el envío de pedidos para mejorar la eficiencia y eficacia del servicio.

A continuación se mostrará el flujo seguido por un pedido desde su generación en el camarero (toma de notas a clientes), hasta la entrega del msimo al consumidor, una vez se ha terminado de cocinar/preparar. Para ello, en primer lugar procederemos a explicar el rol de administrador del restaurante, ya que será este el encargado de introducir en el sistema los platos que el camarero podrá seleccionar para hacer los pedidos a la cocina o a la barra, así como clasificarlos por categorías, pudiendo gestionar la inclusión o supresión de las mismas en función de las necesidades puntuales del establecimiento.

El rol de administrador de la plataforma estará reservado al gestor del establecimiento, ya que desde nuestra plataforma podrá introducir los productos seleccionados, sus precios y las diferentes opciones de preparación disponibles, además de otros aspectos de la gestión del restaurante que se detallan a continuación:

- Gestión de platos:
    - Inclusión de nuevos o supresión de existentes, incluyendo opción de mantenerlos en modo “no visible”, de manera que el camarero no podrá realizar un pedido de dicho plato, sin eliminarlo por completo del sistema. Esta determinación fue tomada con la idea de cubrir la necesidad de que no se pudiera solicitar un plato que se pudiera haber agotado debido al consumo diario en el establecimiento o aspectos como podría ser la estacionalidad de ciertos alimentos.

    - Cambio del precio del plato.

  - Categorización de platos, donde se podrán añadir y eliminar tantas categorías como se desee, facilitando la visibilidad para el camarero, y pudiendo añadir nuevos platos a dichas categorías. Un ejemplo claro de estas divisiones podría ser “Entrantes”, “Postres” o “Bebidas”.

  - Posibilidad de añadir o suprimir campos de información adicional de los platos de la carta, como son las guarniciones, el sabor, o el punto disponibles, ahorrando tiempo al camarero a la hora de realizar el pedido. Un ejemplo de esto podría ser, para un determinado plato de carne, introducir “medio hecho” y “muy hecho” en el campo “punto”, y “con arroz” y “con papas” en el campo “guarnición”, resultando en casillas seleccionables para el camarero (checkboxes) en su interfaz de pedido.

- Gestión de empleados:
  - Posibilidad de añadir o eliminar empleados de la lista de usuarios de la aplicación, con la posibilidad, como ya se viera en el caso de los platos, de mantenerlos “no visibles” si se deseara. Además se da la posibilidad al administrador de poder cambiar de forma sencilla, con un selector, el rol del empleado, haciendo que este, a la hora de acceder a la plataforma desde su dispositivo, sólo le sea visible la interfaz dedicada al rol asignado por el administrador.

Para describir el sistema de manera general, utilizaremos el siguiente diagrama:

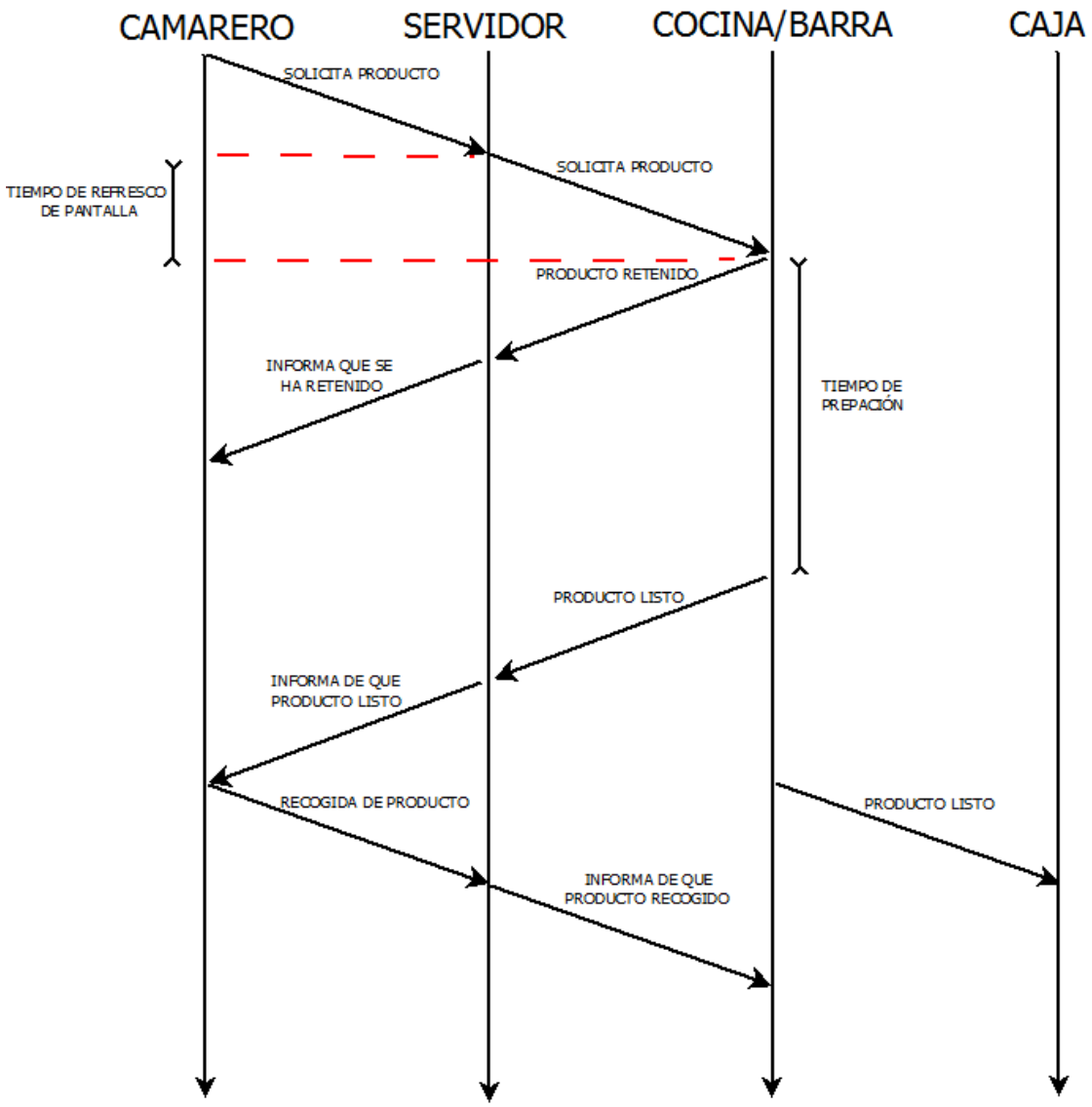

Figura 3-1-1. Diagrama de funcionamiento general del sistema

Como se puede ver en el diagrama, el flujo de la aplicación será iniciado por el camarero, quien solicitará a la cocina o barra los platos pedidos por los clientes. En este aspecto, las colas de cocina y barra se comportan de manera similar, con la salvedad, de que el perfil de barra se ha incluido para recibir pedidos de bebidas, como agua o refrescos. Es por esto que, en función de lo que se pida, la orden se solicitará a una cola u otra.

Después de que el pedido pase por el servidor, el sistema actualizará la pantalla de la cola correspondiente, para lo que se ha establecido un tiempo de refresco de 3 segundos, suficiente para mantener un sistema rápido, sin sobrecarga del servidor.

Una vez en la cola correspondiente, tanto la cocina como la barra deberán retener el pedido mientras se prepara. Un pedido que haya sido retenido dejará de ser cancelable por el camarero.

De esta manera, evitamos que puedan darse situaciones en las que el pedido pueda ser cancelado por el camarero una vez está en preparación, lo que implicaría pérdidas para el establecimiento.

Tras esto, y una vez se haya completado la preparación del pedido por parte de la cocina o la barra, se le indica al camarero dicha situación para que proceda a recogerlos y entregarlos a las mesas de los clientes. Además, y como vemos en el gráfico, se indica al perfil de caja que el producto en cuestión ya ha sido completado, por lo que se procede a su anotación en la base de datos, para la posterior generación de la cuenta.

CAPÍTULO 4. Descripción detallada

En este cuarto capítulo, se mostrará en detalle el flujo de uso de cada rol de usuario ya definido, lo que nos servirá para profundizar sobre cómo se comporta la aplicación y qué debe esperar el usuario.

4.1 Rol de camarero

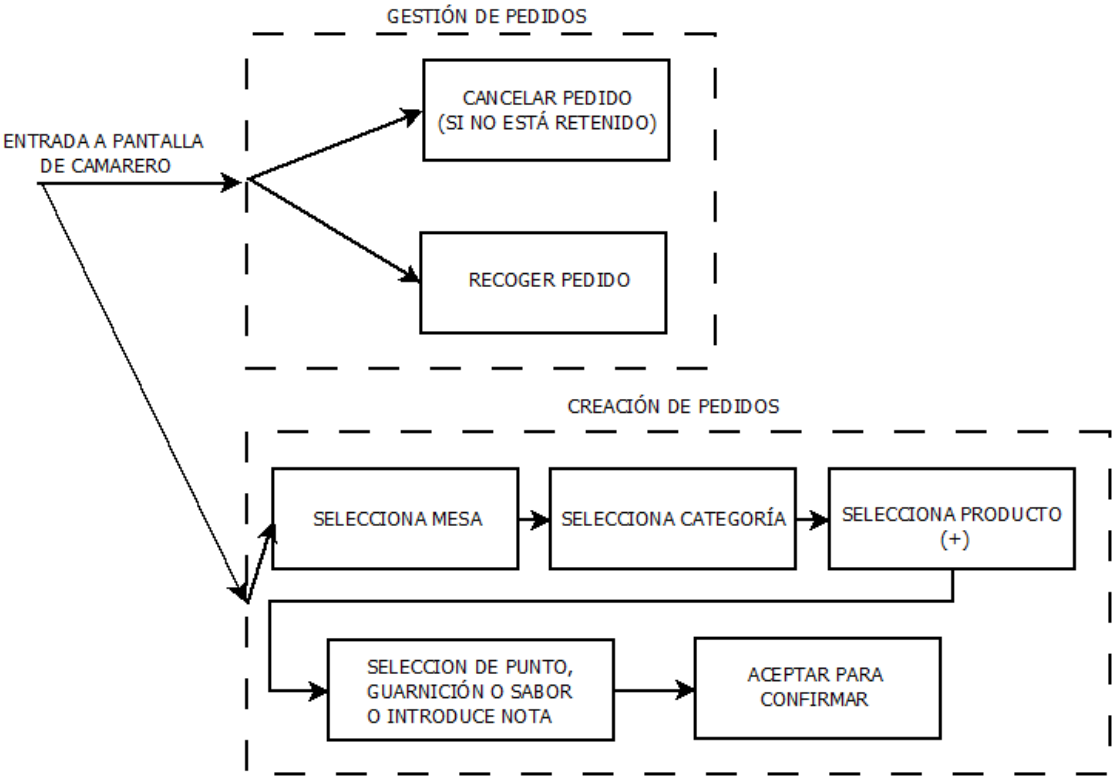

Figura 4-1-1. Diagrama de flujo del camarero

Como podemos ver en el diagrama, el usuario camarero al entrar en su pantalla (seleccionando su nombre en la pantalla inicial de la aplicación), podrá llevar a cabo dos funciones. En la primera, en el bloque superior, referente a los pedidos en curso, la pantalla le mostrará el estado actual de los platos solicitados. Además, desde este apartado, el usuario podrá aceptar un pedido que ya haya sido completado por parte de la cocina o la barra (receptores de las comandas), para retirarlo y entregarlo a la mesa correspondiente. Desde aquí también podrá cancelar pedidos en curso en caso de que fuera necesario, teniendo en cuenta que puede estar ya en preparación, cuando, como ya se ha comentado, no podrán ser cancelados.

Por otro lado, la aplicación le muestra al usuario los pedidos pendientes separados por mesa para mayor eficiencia del servicio, presentándolos en forma de pestañas que pueden ser cerradas y abiertas por el usuario para evitar la acumulación de elementos en pantalla. Esta pantalla, al igual que ocurre en las

pantallas de cocina/barra y caja, se autorefrescará cada cierto tiempo para poder ofrecer siempre el estado real de las peticiones.

Figura 4-1-2. Pantalla de gestión de pedidos del camarero (pestañas cerradas)

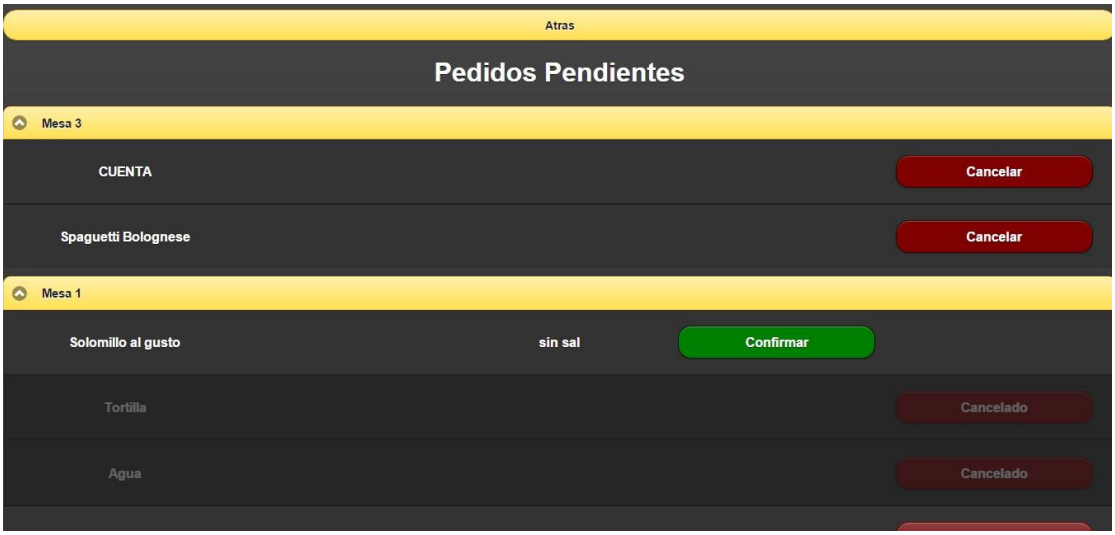

Figura 4-1-3. Pantalla de gestión de pedidos del camarero (pestañas abiertas)

Este diagrama nos indica también la segunda de las funciones que el camarero podrá realizar mediante el uso de nuestra aplicación, como es la generación de pedidos a la cocina. Para ello, y desde la pantalla inicial del camarero podremos seleccionar la mesa para la que queremos realizar los pedidos, pasando, tras esto, a una nueva pantalla donde se nos mostrarán las distintas categorías de productos

disponibles, dispuestas a modo de menús desplegables para mayor comodidad y eficiencia de uso.

Figura 4-1-4. Pantalla de creación de pedidos del camarero (pestañas cerradas)

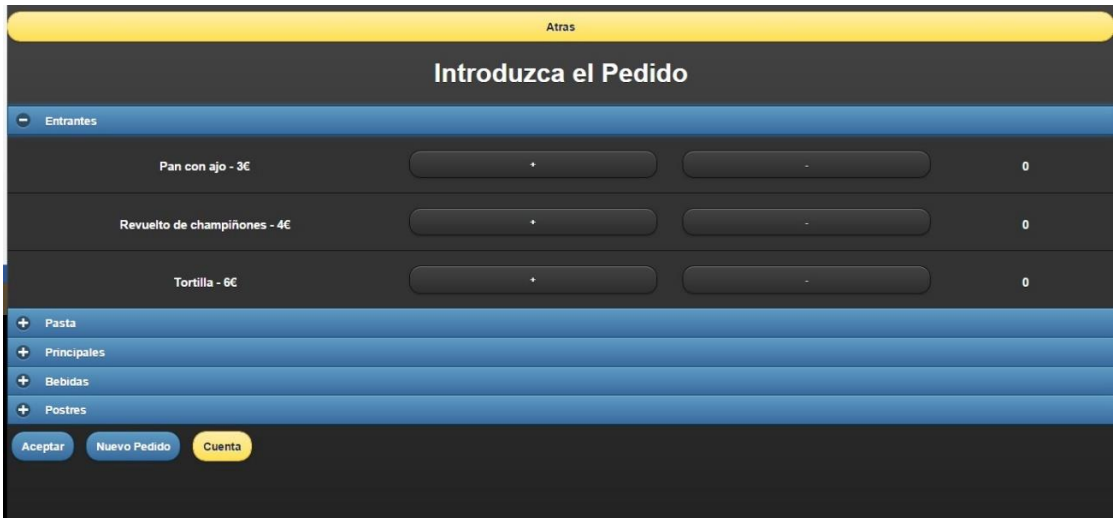

Figura 4-1-5. Pantalla de creación de pedidos del camarero (pestañas abiertas)

Tras la selección de categoría se desplegarán los platos pertenecientes a la categoría seleccionada (figura 4-5), con un botón (+) con el que se iniciará el pedido del producto en cuestión. Una vez pulsado dicho botón, se desplegará un menú modal (popup) en el que se podrá seleccionar el punto, la guarnición o el sabor del producto pedido, mediante checkboxes. Dichas variantes serán las que se hayan introducido desde el panel de administración para dicho producto. En este apartado cabe destacar el carácter multiselección que tendrá la sección de guarnición, mediante lo que se podrá seleccionar diferentes complementos para el plato.

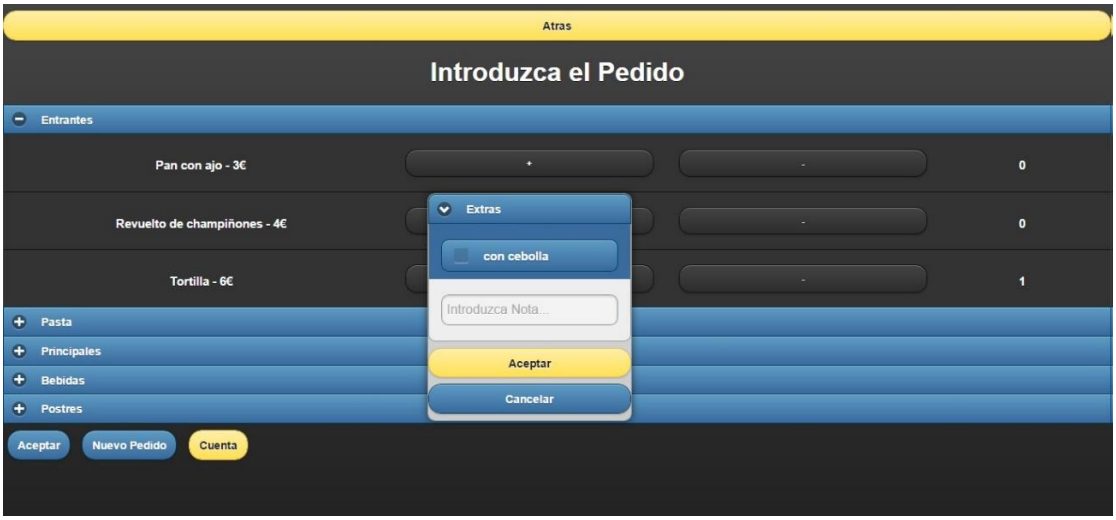

Figura 4-1-6. Pantalla de creación de pedido con menú modal

Además, se añade la posibilidad de que el camarero introduzca una nota dirigida al cocinero en la que podrá indicar cualquier circunstancia especial que pueda no estar contemplada en la generación del pedido, como podría ser la indicación de no usar ciertos alimentos debido a alergias del cliente. Tras completar las casillas deseadas del menú modal, se aceptará el pedido, que irá directamente a la cocina o a la barra en caso de que se trate de bebidas.

Por último, y desde la pantalla de generación de pedidos, se ha incluido la opción de solicitar la cuenta para la mesa seleccionada, que será tratada como un plato normal, a efectos de confirmar su recogida en la sección de pedidos pendientes, una vez que la caja haya generado dicha factura.

4.2 Rol de cocina/barra

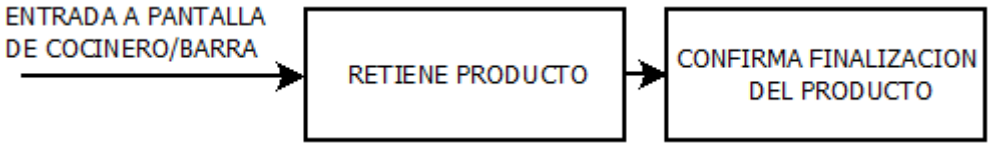

Figura 4-2-1. Diagrama de flujo del cocinero

En el diagrama de uso del cocinero o barra, que recordamos, se comportan de la misma manera, con la salvedad de que se ha impuesto que las bebidas irán a la barra, vemos como las dos únicas funciones que se realizarán serán la retención del plato, con lo que quedará bloqueado para que el camarero no pueda cancelarlo, y la confirmación del plato, una vez este esté terminado y listo para que el camarero lo retire de la cocina para llevarlo a la mesa correspondiente.

Al pulsar el botón retener, este cambiará de estado, mostrando a continuación el texto “Confirmar”, con el que el cocinero dará por terminado el plato solicitado y quedará marcado para el camarero como pendiente de recogida.

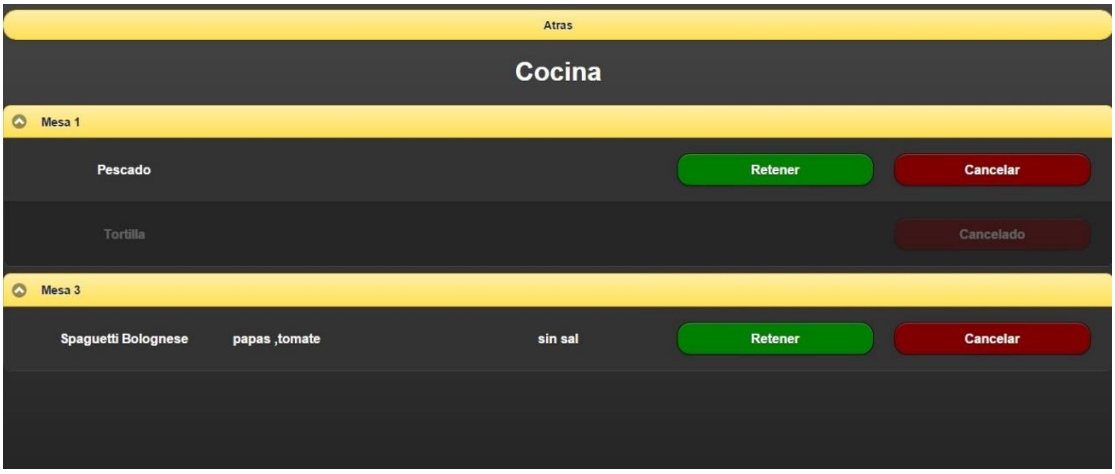

Figura 4-2-2. Pantalla del cocinero

4.3 Rol de caja

En el rol de caja, el usuario se encargará de generar la cuenta, de la forma tradicional, pero podrá utilizar el sistema desarrollado para notificar al camarero(s) que ya se ha preparado y que, como si de un pedido de cocina o barra se tratara, ya está listo para su recogida.

Para una explicación más dinámica, nos apoyaremos en el siguiente diagrama, en el que veremos la evolución del pedido de la cuenta en cada instante, y su movimiento entre la cola del camarero y la de la caja, que nos servirá de base para la comprensión del flujo seguido.

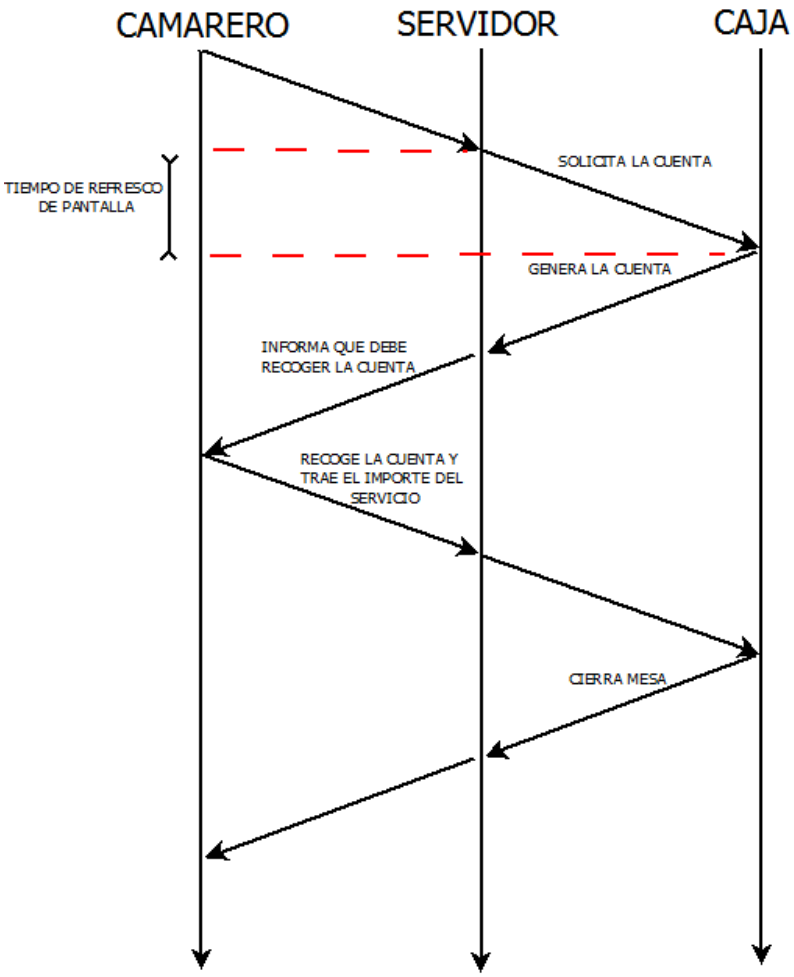

Figura 4-3-1. Diagrama de solicitud de la cuenta en el sistema

Como se puede observar, el flujo vendrá iniciado por la solicitud de cuenta por parte del camarero, generando un pedido pendiente en la pantalla de la caja, para que esta genere la nota. Dicha solicitud se realizará desde la pantalla de generación de pedidos, como si de un plato normal se tratase, pero con un botón dedicado, situado al acabar la lista de categorías de platos, como se puede observar en la figura 4-1-4.

En este rol de caja, se le mostrará al usuario el listado de solicitudes de cuentas que han realizado los camareros y los platos que cada mesa ha ido consumiendo durante la estancia en el establecimiento.

Cabe notar aquí, que sólo se pasan a esta lista de pedidos atendidos, los pedidos que hayan sido marcados como confirmados y entregados, de manera que los pedidos que se hayan podido quedar sin hacer por alguna razón, no pasarán a ser cobrados.

Figura 4-3-2. Pantalla de caja

Una vez confirmemos la cuenta, de la misma forma que el cocinero confirma que un plato ya se ha terminado de hacer, al camarero se le presentará en su pantalla de pedidos pendientes dicha cuenta para que confirme su recogida para entrega al cliente. En la cuenta que recibirá el camarero en su pantalla de pendientes de recogida se le indicará, además, el importe de la misma.

Por último, y tras recibir confirmación del camarero de la entrega de la cuenta, el botón que aparecerá en el escritorio de caja cambiará de manera que al pulsarlo (y confirmar en el menú popup) cerraremos la mesa, confirmando de esta manera el cobro y eliminando los platos de dicha mesa para el siguiente servicio.

Figura 4-3-3. Pantalla de caja indicando el cierre de mesa

4.4 Rol de administrador

El administrador del restaurante será el encargado de introducir los datos necesarios para el correcto funcionamiento del mismo, desde el precio de los platos, hasta los empleados del establecimiento.

Basándonos en la figura 4-4-1, empezaremos a explicar las funciones a realizar por el administrador:

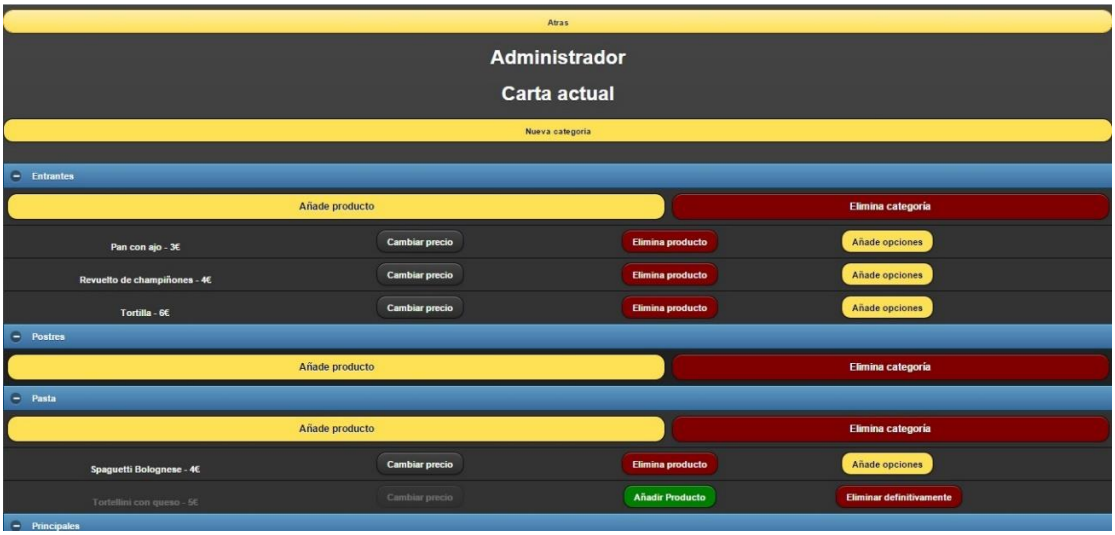

Figura 4-4-1. Pantalla del administrador mostrando la carta actual

Indicar, en primer lugar, el carácter desplegable de las pestañas de las categorías de platos (en azul) con el objetivo de, como bien ocurría en el resto de pantallas anteriormente explicadas, poder cerrar dichas pestañas en caso de que se llegara a un alto número de productos a la vista.

Tras esto, debajo del encabezado “Carta actual” observamos un botón de “Nueva categoría”, con el que añadiremos nuevas categorías de platos, dentro de las que posteriormente se introducirá cada uno de los mismos, con la posibilidad, evidentemente, de cambiar el precio, establecer opciones de selección (sabor, guarnición y punto) o eliminar el plato.

Con el pulsado de dicho botón, el sistema nos mostrará un cuadro de texto donde introducir el nombre de la nueva categoría a introducir, y un botón de aceptar, para que sea guardada en base de datos y visible a los camareros.

Figura 4-4-2. Pantalla del administrador mostrando el cuadro de texto para añadir nueva categoría de producto

Tras esto, pasamos a comentar la funcionalidad de añadir platos nuevos a una categoría, para lo que se ha habilitado un botón bajo la cabecera de la categoría, como bien se puede observar en la figura 4-4-2. Al pulsar dicho botón, se nos mostrará dos cuadros de texto nuevos con los que introduciremos el nombre y el precio del nuevo plato.

Figura 4-4-3. Pantalla del administrador mostrando el cuadro de texto para añadir nuevo producto

Otra de las funciones que el administrador podrá llevar a cabo desde su pantalla será la eliminación de platos, otorgando la agilidad de poder suprimir un plato de la carta tan pronto como se agote. De esta manera, evitaremos que los camareros puedan solicitar platos agotados en el día.

Para esto, y teniendo en cuenta que puede haber platos de temporada o que simplemente se han agotado durante el servicio, se da la opción de dejar los platos en un estado “no visible” de forma que no puedan ser solicitados por el camarero,

pero que no son eliminados por completo, dejando la posibilidad de volver a ponerlo visible en otro momento.

Como se observa en la siguiente figura, tras pulsar “Elimina producto”, el mismo pasa a un estado “no-visible”, que puede ser revocado al pulsar “Añadir producto” o eliminado por completo del sistema con “Eliminar definitivamente”.

Figura 4-4-4. Pantalla del administrador mostrando los dos estados de visibilidad del producto

Otra de las funciones que podrá realizar el administrador será la de cambiar el precio de los platos que son visibles. Pulsando en el botón de “Cambiar precio”, éste cambiará su apariencia para mostrar un botón de confirmación, y el botón de “Elimina producto” pasará a ser un campo de texto donde se introducirá el nuevo importe. Una vez actualicemos el valor y pulsemos “Confirmar precio”, volveremos al estado inicial del botón, con el precio cambiado.

Figura 4-4-5. Pantalla del administrador mostrando el campo de texto para cambiar el precio.

Además, se da la posibilidad de modificar las opciones de preparación de un plato. Esto lo haremos con el botón “Añade opciones”, que nos mostrará un menú emergente (popup) con el que añadir o eliminar opciones de sabor, punto y guarnición de cada plato.

Figura 4-4-6. Pantalla del administrador mostrando menú emergente para la modificación de las opciones del producto.

Como se puede observar en la figura 4-4-6, dispondremos de tres cuadros de texto donde introducir el punto, la guarnición o el sabor del plato seleccionado, quedando añadidos tras el pulsado del botón “Guardar”, y eliminando opciones ya añadidas con el botón “Eliminar”. Para salir de este menú, bastará con pulsar aceptar o pulsar fuera del mismo, volviendo a la vista general del administrador.

Tras comentar el apartado de gestión de la carta y las diferentes opciones para modificar las categorías y platos, pasamos a la siguiente sección de esta pantalla, que encontramos justo debajo, en la pantalla del administrador, bajo el encabezado “Trabajadores actuales”. Será aquí donde se llevará a cabo la getión de empleados del establecimiento.

Figura 4-4-7. Pantalla del administrador mostrando la sección de gestión de empleados.

Como se puede observar en la figura 4-4-7, se ha dispuesto una lista similar a la de los platos, en la que incluimos un botón “Añade empleado” con el que introduciremos un nuevo empleado en el sistema, de forma análoga a como lo hacíamos con los platos dentro de las diferentes categorías, incluyéndose este nuevo empleado al final de la lista, y apareciendo un cuadro de texto donde introducir el nombre del mismo y un menú desplegable en el que aparecerán los roles que podemos asignar al nuevo empleado (figura 4-4-8).

Figura 4-4-8. Pantalla del administrador mostrando el campo donde introducir el nuevo empleado.

Por último, y al igual que ocurría en la sección de platos, podremos fácilmente eliminar a los empleados de nuestro sistema, pulsando el botón “Elimina empleado”, pasando este a un estado “no-visible” de la misma forma que pasaba con los platos que eran eliminados, pudiendo volver a ser activados.

Figura 4-4-9. Pantalla del administrador mostrando un empleado en estado eliminado/no-visible.

CAPÍTULO 5. Base de datos

En este quinto capítulo, explicaremos en profundidad la base de datos que se ha diseñado para nuestro sistema, acompañado de los pertinentes diagramas de bloques en los que se podrá comprobar las relaciones entre las diferentes tablas implementadas.

Además, explicaremos de qué forma realizaremos la conexión de nuestro sistema con la base de datos de Parse en la nube.

Como ya se comentó en el apartado 2.5, la principal característica de Parse es que se trata de una base de datos no relacional, es decir, cualquier relación entre objetos que debamos usar en nuestro sistema, deberá ser creada a partir del uso de peticiones independientes a la base de datos, realizadas utilizando el identificador único del objeto en la base de datos. Por esto, para obtener ciertos datos, se deberán hacer varias peticiones asíncronas, ya que para obtener el objeto que contenga nuestro dato, muchas veces necesitaremos sacar el id de un objeto solicitado previamente.

En el apartado 5.2 se comentará como se ha realizado la conexión con la base de datos, a través de la API de Parse en Javascript, pero antes procederemos a mostrar el diseño de la base de datos que se ha llevado a cabo, de manera que la explicación de dicho apartado se hará más sencilla para el lector.

5.1 Diseño de la base de datos

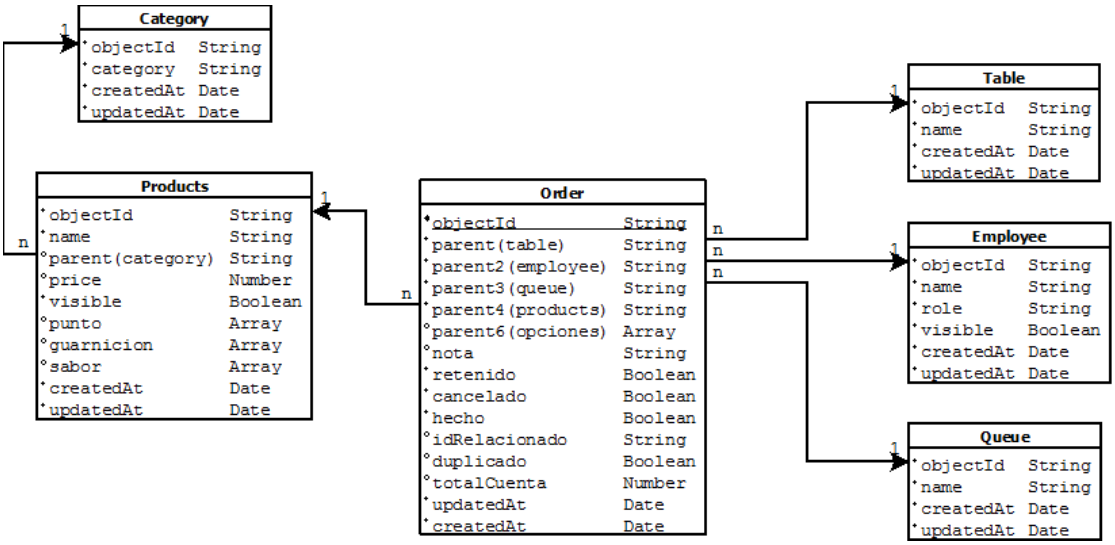

Figura 5-1-1. Diagrama de la base de datos.

Como vemos en el diagrama, el objeto principal será el tipo Order (Pedido), ya que será el que tenga las relaciones hacia los diferentes objetos de la base de datos, siendo estas:

- Clase Table (Mesa): Almacena el nombre de la mesa para la que se realiza el pedido.

- Clase Employee (Empleado): Donde vendrá la información referente al empleado que realizó el pedido, como su nombre o su rol, además de un flag "visible" que indica si el usuario ha sido eliminado por el administrador.

- Clase Queue (Cola): Incluye el nombre de la cola en la que se encuentra el pedido en cada momento, ya que los productos irán cambiando de cola según vayan completándose las tareas Pedido-Preparación-Entrega-Pago. Los posibles valores de la cola, y que además no son modificables por el usuario, son "Camarero", "Barra", "Cocina" y "Caja".

- Clase Products (Producto): Objeto que llevará la información relevante del producto solicitado, como su precio, nombre, opciones de preparación (sabor, guarnición y punto) y el estado de visibilidad, donde, como ya se ha descrito, se indica si el plato está eliminado o no. Además, se incluye una relación con la clase Category (Categoría), donde se indicará la categoría a la que pertenece el plato.

Por otro lado, la clase Order, también incluye otros campos necesarios para el funcionamiento del sistema:

- Opciones: Campo que incluirá las opciones de preparación que el camarero haya seleccionado a la hora de generar el pedido mediante los checkboxes.

- Nota: Donde irá el texto dirigido al cocinero introducido por el camarero.

- Retenido: Flag utilizado para indicar que el pedido ha sido retenido por el cocinero, con lo que no podrá ser cancelado.

- Cancelado: Flag utilizado para indicar que el pedido ha sido cancelado, ya sea por parte del camarero o el cocinero.

- Hecho: Flag que indica que la preparación ha sido completada.

- Duplicado: Flag utilizado para indicar si se trata de un pedido duplicado. Un pedido se duplica para mostrarlo en la pantalla del camarero y el cocinero al mismo tiempo. Estos dos pedidos serán idénticos pero con distinti valor de Queue (cola).

- IdRelacionado: Campo que almacena el identificador del pedido relacionado. Dicho pedido relacionado es el duplicado del que hablamos en el campo. De esta manera podremos tener acceso a ambos pedidos cuando necesitemos cambiar algún flag.

- TotalCuenta: Campo que se utiliza para rellenarlo con el total de la cuenta, una vez que se ha terminado el servicio a una de las mesas, aunque es un campo meramente informativo, ya que, como se ha visto, al solicitar la cuenta, el camarero verá generado un pedido pendiente, con el que deberá recoger la cuenta desde la caja al estilo tradicional.

5.2 Conexión con la base de datos en la nube

Como se comentó al inicio de este capítulo, las peticiones a la base de datos se llevarán a cabo de forma asíncrona, con lo que se deberán utilizar estructuras de código *success-error* con las que controlaremos que una petición al servidor Parse haya sido o no completada.

Estas estructuras nos permiten controlar el flujo que debe seguir nuestra aplicación, frente a una petición de datos remota, en función de si dicha llamada se ha completado con éxito (*success*) o ha fallado (*error*). Es importante tener en cuenta esta forma de actuar a la hora de programar sucesivas tareas que puedan requerir datos de peticiones anteriores, para lo que deberemos anidar dichas peticiones en varias estructuras *success-error,* como se puede ver en la figura siguiente:

```
query.get(idProducto,{
        success: function(obj){
            //SOLICITO EL PRODUCTO SELECCIONADO, PARA MARCARLE
            //EL FLAG DE RETENIDO Y GUARDARLO
            obj.set("retenido",true);
            obj.save();

            //HAGO LA PETICION DEL DUPLICADO DEL PRODUCTO SELECCIONADO
            var myOrder = Parse.Object.extend("order");
            var query = new Parse.Query(myOrder);
            query.equalTo("duplicado",true);
            query.equalTo("idrelacionado",obj.id);
            query.find({
                success: function(res){
                    //UNA VEZ OBTENIDO EL DUPLICADO, MARCO EL FLAG EN ESTE
                    res[0].set("retenido",true);
                    res[0].save();
                },
                error: function(){
                    //AQUÍ DEFINIMOS LAS ACCIONES A REALIZAR EN CASO DE
                    //FALLO DE LA SEGUNDA PETICIÓN
                    alert("No encontro duplicado en manejaConfirmaOrder");
                }
            })
        }
    },
    error: function(error) {
        //AQUÍ DEFINIMOS LAS ACCIONES A REALIZAR EN CASO DE
        //FALLO DE LA SEGUNDA PETICIÓN
        alert("Error: " + error.code + " " + error.message);
    }
})
```

Figura 5-2-1. Código de peticiones enlazadas.

Esto, unido a que no se trata de una base de datos relacional, en la que podríamos traernos un objeto y sus objetos relacionados con una sola petición, nos obliga a realizar varias peticiones para obtener los datos necesarios que se encuentran en otros objetos. Estas peticiones se realizarán tomando los identificadores únicos de los objetos relacionados y haciendo una llamada independiente.

Un ejemplo de cómo actuar podría ser el siguiente:

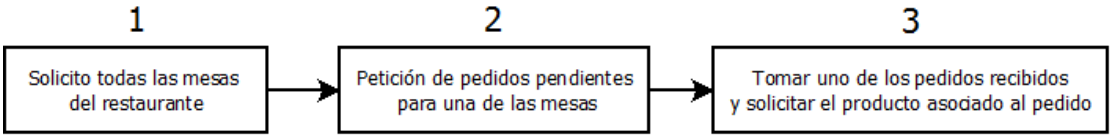

Figura 5-2-2. Diagrama de peticiones enlazadas.

El primer paso sería hacer una solicitud de las mesas disponibles en el sistema, previamente introducidas por el administrador en su pantalla. A partir de ellas, tomaríamos una y, usando el identificador único del objeto tipo “mesa”, realizar una petición de los pedidos pendientes de esa mesa, obteniendo un array de objetos tipo “pedido” que llevará dentro el identificador único del producto solicitado. Por último, y tomando dicho identificador de producto, procederíamos a la petición del objeto “producto”, recibiendo finalmente (paso 3 de la figura 5-2-1), el plato solicitado para la mesa en cuestión, donde tendremos, entre otros datos, el nombre del plato y el precio.

En nuestro proyecto, al usar la API de Parse para conectarnos con la base de datos remota, nos abstraeremos en gran medida de las funciones que se realizan para realizar dichas conexiones de manera satisfactoria, limitándonos a utilizar las funciones optimizadas proporcionadas para la gestión de las peticiones asíncronas.

Para poder conectarnos con la base de datos, necesitaremos obtener un identificador único de aplicación con el que enlazar nuestra aplicación con la base de datos en los servidores de Parse, para lo que tendremos que registrarnos en la web de parse (*parse.com*) y crear una aplicación, con la que obtendremos nuestras credenciales.[14]

Una vez hecho esto, simplemente usando la línea de código, en JavaScript:

```
Parse.initialize("KZyQQPDxsIjDluznhwdTn02n5vUgin3gdFCfxBO8", "6FXIp0kZOGSZohrsLzHKe7UsDXwBDYoi5U1p07vG");
```

Conseguiremos que nuestro sistema quede registrado en parse, sin tener que realizar ninguna tarea adicional como podría ser renovar las credenciales de acceso, ya que todo esto vendrá gestionado por la API de Parse.

Parse nos ofrece, además una consola de control desde su web, con la que, al introducir nuestras credenciales, accederemos, entre otras cosas, a una pantalla donde gestionar y controlar nuestra base de datos, siendo esto bastante útil para comprobar que todas nuestras acciones realizadas en la aplicación se hayan realizado correctamente.

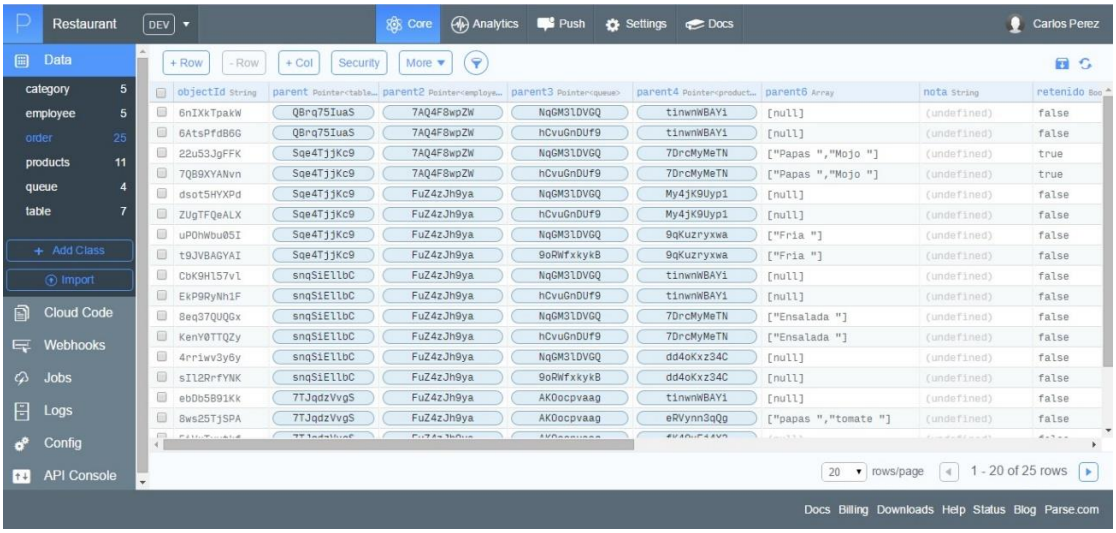

Figura 5-2-3. Consola de Parse.

CAPÍTULO 6. Implementación modular

En este capítulo mostraremos como se implementa el sistema detallado en el capítulo 4, pero profundizando técnicamente.

En este capítulo explicaremos cómo se han implementado los diferentes roles de usuario de la aplicación. Para ello será necesario el uso de diagramas de bloques con los que describir las funciones que realizan dicha implementación. Estos bloques, además nos ayudarán a la comprensión del funcionamiento global, al utilizarse con diferentes roles.

Antes de proceder a la descripción de los bloques, se hará una pequeña introducción en la que detallaremos la estructura utilizada en el documento HTML, que será el que nos sirva como “esqueleto” de la aplicación, al estar basada, esencialmente, en *JavaScript*.

6.1 Estructura HTML de la pantalla inicial

Como se describió en el segundo capítulo, cuando introducíamos el concepto de HTML, el fichero HTML será el que contenga la estructura de la aplicación, y será donde, una vez cargado el fichero JavaScript, se realicen las modificaciones para que la aplicación sea funcional.

En nuestro documento HTML se han creado varias divisiones ( ) que harán las veces de pantallas independientes de la aplicación, es decir, cada una de esas divisiones corresponderá a una pantalla, dejando la navegación entre pantallas relegada al fichero JavaScript. Esto es posible gracias al uso de una librería particular de jQuery, llamada jQuery Mobile, que incluyendo el atributo “data-role=page”, hará que cada uno de esas divisiones sea una pantalla independiente de la aplicación. De no haber usado dicha librería, lo que veríamos serían todas las pantallas del sistema unas debajo de otras, sin división entre ellas.[15]

Para explicar mejor esto, se recurre a la siguiente imagen, donde mostramos los bloques del documento que corresponden a las diferentes pantallas:

Figura 6-1-1. Estructura HTML y su correspondencia con las pantallas.

Como vemos en la figura 6-1-1, cada división del documento HTML corresponderá a una pantalla de la aplicación, mostrándose sólo la división que corresponda en cada momento, según el flujo de la misma. Indicar aquí que se han agrupado las páginas de barra, caja y cocina por ser similares.

Dentro de estas estructuras básicas, comenzaremos a colocar jerárquicamente lo elementos necesarios para cada pantalla. En primer lugar, la pantalla de inicio, en la que el usuario selecciona su rol pulsando su nombre en la lista, únicamente necesitaremos colocar las divisiones de cada rol de usuario, es decir, camareros, cocineros y barra, reservando para los roles de caja y administrador botones simples aislados, ya que son roles normalmente reservados a un único usuario.

La estructura interna de esta primera pantalla del sistema quedaría así:

```
<div data-role="page" id="main" data-theme="a">
   <h1 data-theme="b" style="text-align:center;">Indique su usuario</h1>
   <div data-role="collapsible-set" data-theme="b" data-collapsed-icon="arrow-d" data-expanded-icon="arrow-u">

      <div data-role="collapsible" data-content-theme="a">
         <h3 >Camareros</h3>
         <ul id=w data-role="listview">
         </ul>
      </div>
      <div data-role="collapsible" data-content-theme="a">
         <h3 >Cocineros</h3>
         <ul id=c data-role="listview">
         </ul>
      </div>
      <div data-role="collapsible" data-content-theme="a">
         <h3 >Barra</h3>
         <ul id=b data-role="listview">
         </ul>
      </div>

      <a href="#cash" onclick='seleccionaUsuario("#sh")' data-role="button" data-theme="e">Caja</a>
      <a href="#admin" onclick='seleccionaUsuario("#adm")' data-role="button" data-theme="e">Administrador</a>
   </div>
</div>
```

Figura 6-1-2. Estructura HTML de la pantalla inicial

En la figura se ha denotado con colores los bloques anteriormente comentados, reservados a los roles de camarero, cocinero y barra, dentro de los cuales se incluirán elementos tipo lista (<li>) con el nombre del usuario. Dichos elementos serán introducidos por el fichero JavaScript una vez se haya lanzado la página, procediendo a hacer la petición a Parse de los empleados del restaurante, y colocándolos según el rol asignado desde la consola de administrador, ya que, recordemos, la clase Employee (empleado) contiene el campo “role” con el rol asignado.

Además, el uso de la hoja de estilos propia de jQuery Mobile dotará la página de una apariencia más atractiva, simplemente con el uso de unos pocos atributos en los elementos HTML, como clases o “data-roles” [16], obteniendo un efecto como el siguiente:

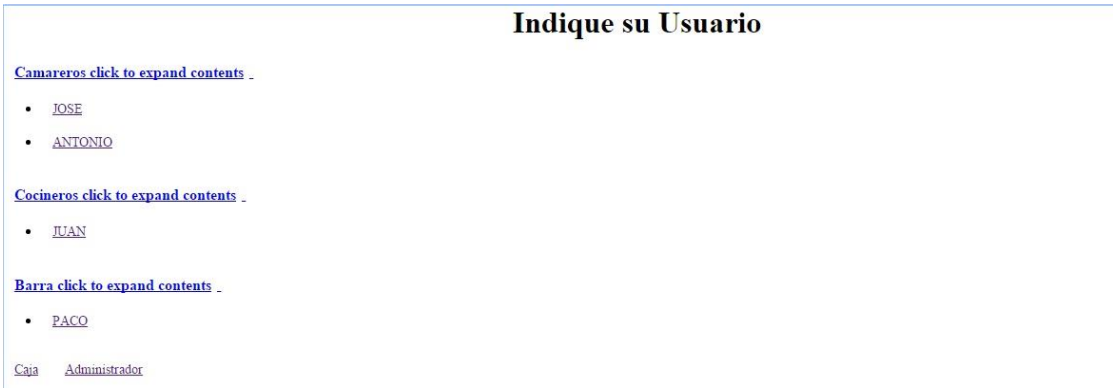

Figura 6-1-3. Apariencia de la pantalla inicial sin CSS de jQuery Mobile

Figura 6-1-4. Apariencia de la pantalla inicial con CSS de jQuery Mobile

6.2 Implementación de la pantalla de pendientes

En este apartado explicaremos como se generan las pantallas de pedidos pendientes de los tres tipos de roles presentes en el sistema, ya que funcionan de forma similar, variando únicamente en las particularidades de cada usuario, pero usando la misma estructura de peticiones al servidor. Para ello se mostrará el funcionamiento de la pantalla del camarero, donde se verán los pedidos pendientes de recogida del camarero, asi como los enlaces a las diferentes mesas, con los que realizar nuevos pedidos para las mismas.

Con esto, y de la misma forma que ocurría en la pantalla inicial, utilizaremos una estructura HTML que “rellenar” con el uso de jQuery (JavaScript). En este caso, el único código HTML que necesitaremos serán unas cabeceras que nos sirvan de

guía para incluir el grueso de la información traída del servidor, y un botón de “atrás” para facilitar la navegación.

```
<div data-role="page" id="table" data-theme="a" >
   <a href="#main" data-role="button" class="botonAtras" data-theme="e">Atrás</a>
   <h1 id="cabecerapend" data-theme="b" style="text-align:center;">Pedidos pendientes</h1>
   <h1 id=t data-theme="b" style="text-align:center;">Nueva comanda, indique la mesa</h1>
</div>
```

Figura 6-2-1. Estructura HTML de la página de camarero

El resto de elementos de esta vista, como ya se indicó, serán incluidos mediante jQuery. Al seleccionar el rol de usuario, en este caso del camarero, se llamará a una función que se encargará de solicitar a la base de datos remota la información de los pedidos pendientes de entrega del camarero y las mesas disponibles en el restaurante para solicitar un nuevo pedido, anexando dichas mesas, en caso de que no hubiesen sido cargadas con anterioridad, y lanzando un proceso de autorrefresco de la pantalla que permitirá seguir en tiempo real el estado de los pedidos lanzados a cocina/barra.

Llegados a este punto, y con el objetivo de facilitar la comprensión del flujo llevado a cabo al abrir un página de camarero, se incluye un pequeño diagrama de flujo sobre el que explicaremos las modificaciones realizadas a nivel de código Javascript que acabarán con la información solicitada presentada en pantalla.

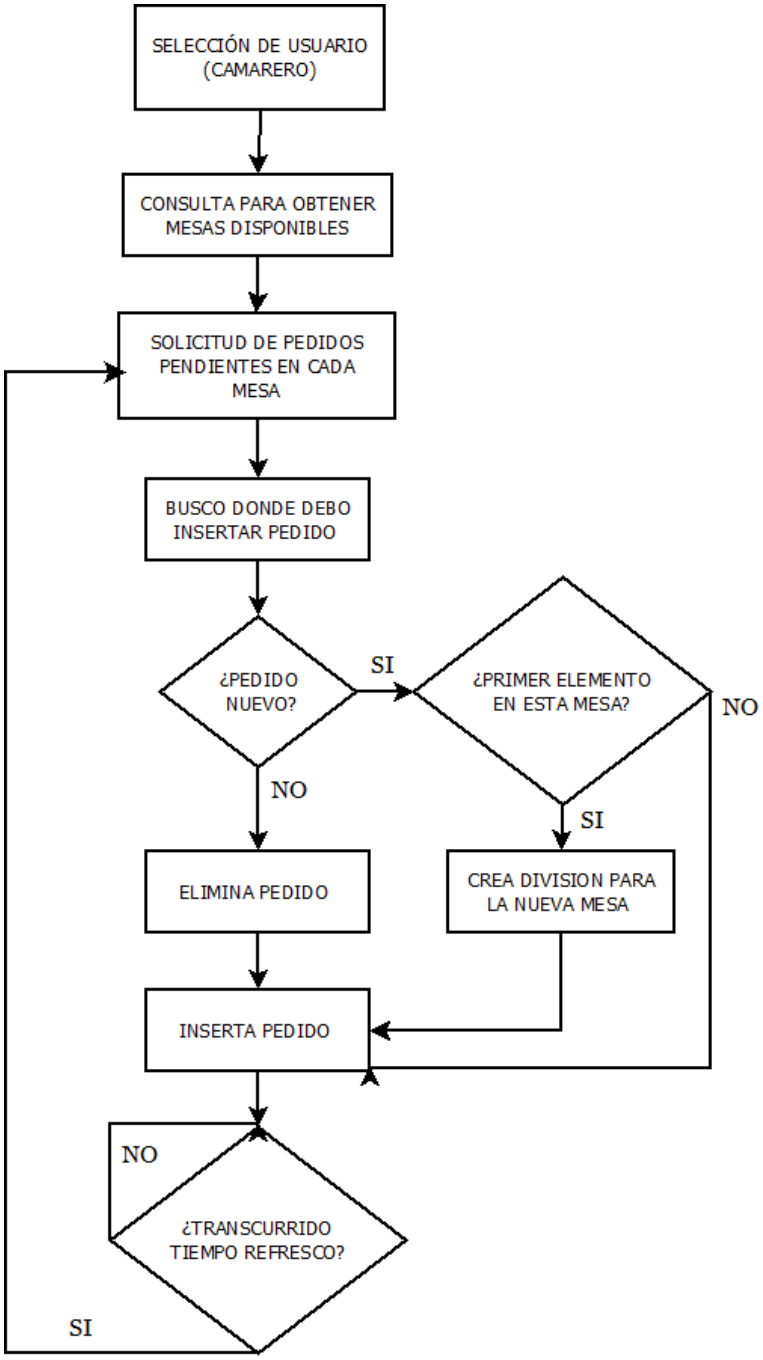

Figura 6-2-2. Diagrama de flujo del refresco de pantalla

En primer lugar, se solicitan a la base de datos remota, las mesas disponibles en el restaurante, para lo que llamaremos a nuestra función *solicitaMesa(),* tras lo que haremos un bucle por las mismas con el objetivo de “pintar” los enlaces a cada una de las mesas (recordamos, los botones azules bajo el encabezado “Nueva comanda, indique la mesa”).

Tras esto, haremos uso de la función *refrescaPagina(idPaginaRefrescar)*, a la que le pasamos como parámetro el id de la página que queremos refrescar (el identificador del contenedor correspondiente al rol seleccionado, de acuerdo a lo comentado en la figura 6-1-1). Esta función será la encargada de realizar un bucle por las mesas con el que haremos las solicitudes a la base de datos de los pedidos pendientes para el rol de camarero y para la mesa en particular. Una vez obtenidos los pedidos de la mesa, se comprueba con la función, *obtieneLugarAEscribir(rolUsuario)*, en que elemento se debe insertar el pedido que se va a mostrar. Notar aquí que dicha función nos dará los elementos donde anexar nuestro nuevo pedido, en función del rol de usuario seleccionado, ya que la función de refresco de pantalla es común para los tres tipos de roles usados en la aplicación.

Una vez obtenido el lugar donde introducir el pedido, comprobamos si el producto contenido en el pedido ha sido alguna vez solicitado a la base de datos, para lo que se ha creado un pequeño array a modo de caché, con el objetivo de minimizar el número de peticiones de un mismo elemento. Dicha comprobación la realizaremos haciendo una llamada a la función *compruebaCache(objetoOrder, enlace, cache),* que recibe como parámetros el pedido a mostrar, el campo del objeto *'Order'* donde se encuentra el identificador del producto (enlace) y el array que sirve como cache, y devolviendo como resultado un booleano indicando si fue encontrado en cache, y la posición en caché en caso positivo.

Después de chequear la caché, se comprueba que dicho pedido no exista ya en pantalla (mediante su identificador único, que se usará como como atributo 'id' en HTML). En caso de que exista, debido a un refresco anterior, se eliminará de pantalla para volver a pintarlo, consiguiendo de esta manera que los indicadores de estado del pedido estén actualizados, ya que podría ocurrir que el cocinero cambie el estado del pedido (cancelarlo, por ejemplo) y el camarero no note dicho cambio. En caso de no existir, se comprueba que la división de la mesa ya esté generada y, en caso contrario, generarlo utilizando la función *creaListviews(mesa, sufix, id_pagina)*. Esta función recibe como parámetros el índice en el array de mesas de la mesa a insertar, además de un sufijo que se usará para generar el identificador del elemento HTML de la mesa. Este sufijo será único para cada rol y vendrá dado

por la función *obtieneLugarAEscribir(rolUsuario)* llamada anteriormente. Por último, *creaListviews()* también recibirá como parámetro el identificador de la página donde se debe empotrar la mesa, dato que también obtuvimos de *obtieneLugarAEscribir().*

Tras haber generado la mesa donde introducir los pedidos, se insertarán los pedidos pendientes de dicha mesa, lo que haremos con las funciones *tomaProductYCantidad()* y *agregaProductyCantidad(),* funciones que realizan el mismo cometido, con la salvedad de que la primera trae los datos de la base de datos, los guarda en caché y los "pinta" en pantalla, y la segunda solamente "pinta". Además, a la hora de realizar la inserción, estas funciones tienen en cuenta el estado del pedido, ya que la apariencia de los elementos cambiará en función de ello, pudiendo quedar el elemento en un estado "disabled" de forma que el camarero no pueda interactuar con el mismo, en el caso de que el pedido esté retenido por la cocina/barra, como ya se vio por ejemplo, en la figura 4-4-2.

Tras esto, el sistema volverá al punto de solicitud de pedidos pendientes, actualizando los elementos necesarios una vez transcurrido un tiempo determinado, establecido en la función *refrescaPagina()*.

6.3 Implementación de la pantalla de nuevos pedidos

En la pantalla de pedidos, usaremos una estructura HTML parecida a la que usábamos en los pedidos pendientes, basándonos en el uso de estructuras desplegables dentro de las que introducíamos listas de elementos (<ul> y <li>).[17] La principal particularidad que tendrá esta pantalla, a nivel visual, será el uso de elementos emergentes (popups) para la inclusión de las opciones de pedido (guarnición, punto y sabor) y para la solicitud de la cuenta.[18]

Una vez más, y como ya hiciéramos en el apartado 6.2, recurriremos a la inclusión del código HTML inicial que iremos rellenando mediante el fichero JavaScript, para realizar una explicación más precisa del proceso seguido.

```
<div data-role="page" id="waiter" data-theme="a">

   <a href="#table" data-role="button" data-theme="e" class="botonAtras" onclick="refrescaPagina(0, 'table')">Atrás</a>
   <h1 data-theme="b" style="text-align:center;">Introduzca el pedido</h1>
   <div id="cabecera_productos" data-role="collapsible-set" data-theme="b">
      <a href="#popupDialogCaja" id=pidecuenta data-rel="popup" data-position-to="window" data-role="button" data-inline="true"  data-theme="e">Cuenta</a>

      <div data-role="popup" id="popupNested" data-theme="b">
         <div data-role="collapsible-set" data-theme="b" data-content-theme="b" data-collapsed-icon="arrow-r" data-expanded-icon="arrow-d" style="margin:0; width:250px;">
            <div data-role="collapsible" data-collapsed="false" data-inset="false">
               <h2 class="toggle-parent">Extras</h2>
               <ul data-theme="c" id=li1 class="listPopup" data-role="listview">
               </ul>
               <ul data-theme="b" id=li2 class="listPopup" data-role="listview">
               </ul>
               <ul data-theme="c" id=li3 class="listPopup" data-role="listview">
               </ul>
               <ul data-theme="c" id=li4 class="listPopup" data-role="listview">
                  <li><input type="text" data-inline="true" placeholder="Introduzca Nota..."></li>
               </ul>
            </div>
            <a href="#" id=aceptapopup data-role="button" data-rel="back" data-theme="e">Aceptar</a>
            <a href="#" id=cancelapopup  data-role="button" data-rel="back" data-theme="b">Cancelar</a>
         </div>
      </div>

      <div data-role="popup" id="popupDialogCaja" data-overlay-theme="a" data-theme="c"
         data-dismissible="false" style="max-width:400px;" class="ui-corner-all">
         <div data-role="header" data-theme="a" class="ui-corner-top">
            <h1 style="margin-right: 10%;margin-left: 10%;">¿Pedir Cuenta?</h1>
         </div>
         <div data-role="content" data-theme="d" class="ui-corner-bottom ui-content">
            <a href="#" data-role="button" data-inline="true" data-rel="back" data-theme="c">Cancelar</a>
            <a href="#" id=cuentadialog data-role="button" data-inline="true" data-rel="back" data-transition="flow" data-theme="b">Aceptar</a>
         </div>
      </div>

   </div>
</div>
```

Figura 6-3-1. Estructura HTML inicial de pantalla de pedidos

Como se muestra en la figura 6-3-1, de nuevo contaremos con una cabecera (“cabecera_productos”) que nos servirá de guía para introducir los pedidos, ya que será el contenedor donde anexaremos las diferentes categorías, introduciendo, por último, los platos de la carta, dentro de estas últimas.

A continuación, siguiendo con el mismo esquema seguido en el apartado 6.2, nos apoyaremos en el uso de un diagrama de flujo para explicar cómo rellenamos dinámicamente el HTML mostrado anteriormente.

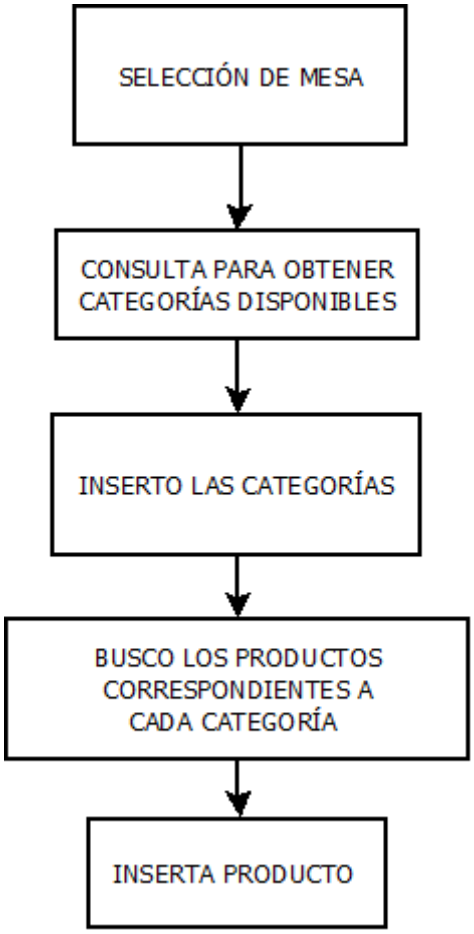

Figura 6-3-2. Diagrama de flujo de carga de categorías

En el diagrama vemos que el flujo a seguir para la carga de categorías y platos es más corto que en la pantalla de pendientes, debido a que esta no requiere refresco constante. Dicha carga, la realizaremos llamando a nuestra función *cargaCategorias(admin)*, que recibirá como parámetro un flag que indicará si la carga se está realizando en la pantalla de creación de pedidos o en la pantalla de administrador, pudiendo de esta manera utilizar dicha función en ambos casos. Esta función tiene en cuenta la posibilidad de que las categorías traídas de la base de datos y las que estén pintadas (procedentes de una carga anterior) no sean las mismas, situación que se dará si el administrador durante el transcurso de un servicio, decidiera eliminar una de las categorías, lo que podría dar lugar a ciertos problemas con pedidos en curso.

Tras esto, procederemos al relleno de las categorías añadidas, para lo que haremos uso de la función *solicitaRestaurante(),* que se encargará de rellenar las

listas creadas para cada categoría con los productos pertenecientes a la misma, realizando un bucle entre las categorías anteriormente obtenidas de base de datos, llamando a la función *rellena()*, encargada de solicitar los productos de la categoría y, tras guardarlo en caché si fuera necesario, pintarlos.

Por otro lado, y siguiendo con la figura 6-3-1, se incluyen los popups nombrados al inicio de esta sección. Denotado en rojo se observa la estructura del popup de las opciones del pedido, donde podemos distinguir las tres listas (<ul>) donde irán introducidas dichas opciones de pedido.

Notar en este punto, que la estrategia seguida para la presentación de las opciones de cada plato en este menú ha sido la de vaciar las listas y rellenarlas al seleccionar un producto, ya que, como podría parecer evidente teniendo en cuenta que el fichero HTML se utiliza como estructura inicial, a priori no podemos saber el número de menús necesarios. De esta manera, se usará siempre el mismo menú, cambiando su contenido cada vez que el camarero seleccione un producto nuevo.

Tras esto, y una vez ya tengamos cargados los platos susceptibles de ser solicitados a cocina/barra, procederemos a describir el flujo seguido para la generación de un nuevo pedido por parte del camarero.

6.4 Implementación del sistema de pedidos

Llegados a este punto, describiremos el proceso seguido para realizar un nuevo pedido por el camarero, al estar ya cargados en pantalla todos los productos seleccionables de la carta del restaurante, como ya se vió en la sección anterior.

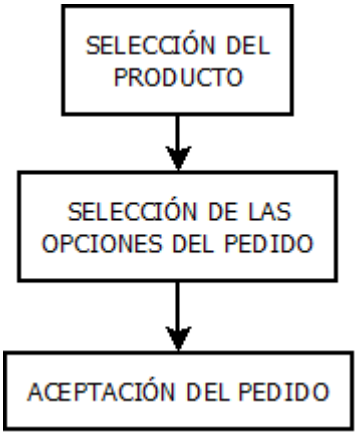

Figura 6-4-1. Diagrama de flujo de creación de pedidos

Como vemos en el diagrama, al seleccionar el plato de la lista, pulsando el botón '+' junto al nombre, se lanzará el menú con las opciones seleccionables para personalizar el pedido. Como ya se comentara en el apartado 6.3, dicho menú será cargado con las opciones de cada plato en el momento que se produzca el pulsado del botón '+'. Dicho botón lanzará la función *creaPopup()* que recibirá el identificador del plato. Con dicho identificador, localizaremos el plato en nuestra caché, para mostrar las posibles opciones del mismo. Antes de esto, como ya se había comentado, se realizará un vaciado de los elementos en las listas de opciones del popup, eliminando de esta manera las opciones correspondientes a otros platos que hayan podido ser solicitados anteriormente, ya que, recordamos, el elemento popup se reutiliza para todos los elementos.

Notar aquí, que para las categorías de opciones "sabor" y "punto" se ha incluido una pequeña rutina con la que evitamos que se puedan seleccionar valores complementarios, como podría ser, en el caso de la carne, que esta pudiese ser seleccionada "muy hecha" y "poco hecha" a la vez.

Una vez vaciado los elementos antiguos y cargados los nuevos, el usuario camarero seleccionará las opciones deseadas o introducirá una nota para el cocinero, en caso que así lo deseara. Tras esto, y una vez pulsado el botón "Aceptar", el sistema procederá a leer las casillas (*checkbox*) seleccionadas, así como la nota, con lo que se llamará a nuestra función *enviaOrder(),* que será la encargada de gestionar a qué cola se enviará el pedido, en función de la categoría

del plato (si es una bebida irá a la barra), diferenciando que el “plato” seleccionado haya sido la cuenta, para lo que se enviará a la cola de la caja, pero que a efectos de solicitud se comportará como un plato más.

Una vez hecha esta distinción, se enviará por último a la función *EnlazaProductoAOrder()*, a la que le pasará como parámetros todos los campos para que genere el objeto clase “Order”, enviándolo a la base de datos y creándole un duplicado que será usado para llevar el control en ambos lados del pedido. Este duplicado será enviado a la cola de pendientes del camarero, de forma que pueda llevar un seguimiento del pedido, pudiendo confirmar su estado en todo momento. Dicho duplicado, como podría parecer lógico, irá enlazado con el pedido inicial, con el que realizaremos los cambios en los flags de preparación cuando sea necesario.

6.5 Implementación del sistema de sincronismo entre pantallas

Una vez explicado como se muestran los elementos dinámicos del sistema y la generación de un pedido, procedemos ahora a comentar cómo se ha implementado el sistema de sincronismo entre pantallas, en el que mantenemos las pantallas de las dos partes (camarero y cocina/barra) actualizadas en función de lo que la otra parte realice, como por ejemplo, cuando el cocinero indica que el pedido ya está listo y el camarero percibe esto con una indicación de que debe recogerlo.

Como ya se ha comentado, las pantallas de pedidos pendientes, se autorefrescarán pasados 3 segundos, de manera que los pedidos mostrados a cada usuario estarán en constante refresco, manteniendo así el estado real de los pedidos en cada momento.

Dicho esto, y fijándonos en la figura 3-1-1, en la que se mostraba el diagrama de interacciones entre los distintos usuarios, mostraremos de una manera más gráfica la apariencia de los botones que se le muestran a los usuarios, según su rol:

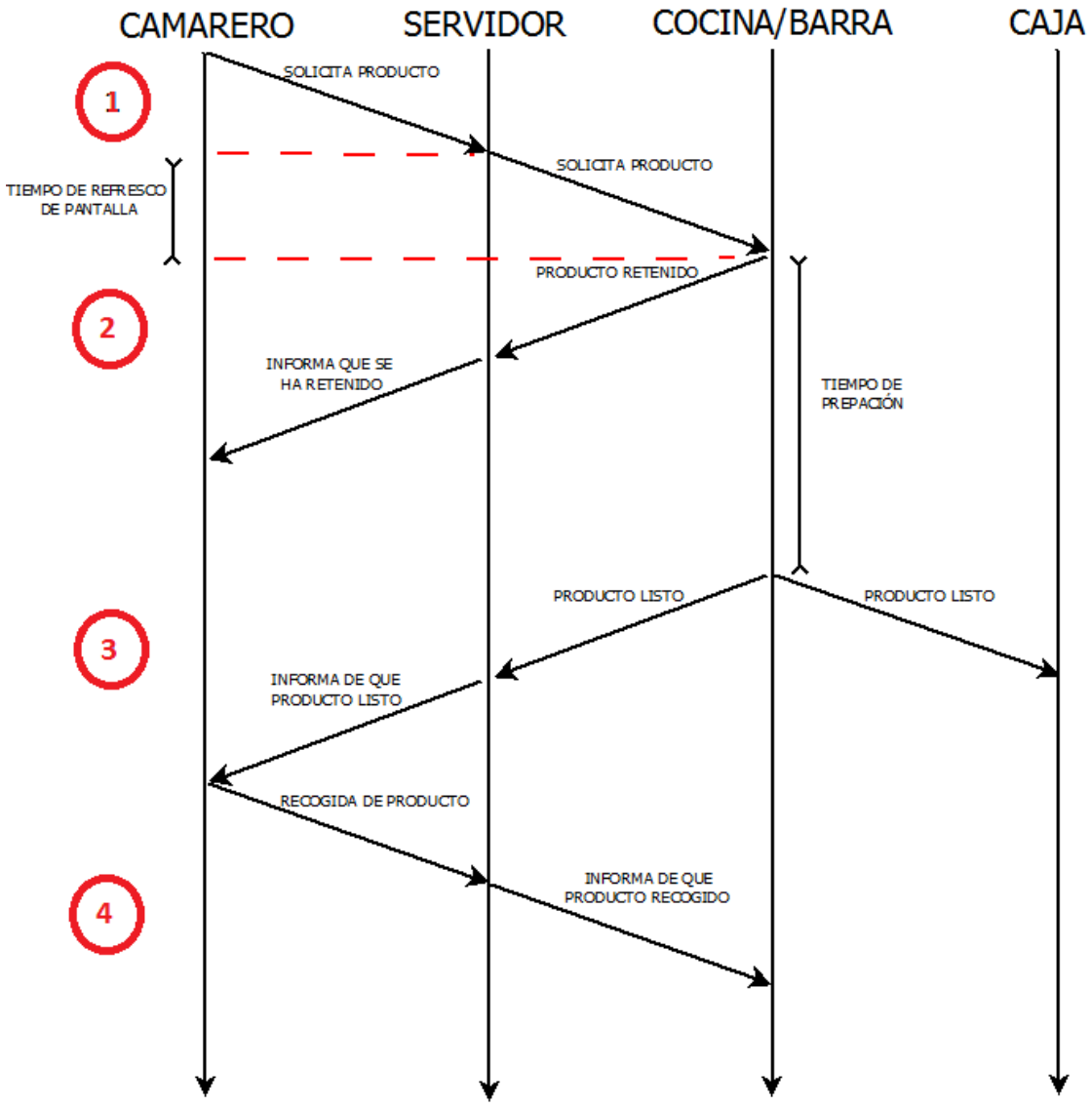

Figura 6-5-1. Diagrama del estado de los botones

Una vez realizado el pedido por parte del camarero, el siguiente ciclo de recarga, nos mostrará en pantalla el elemento recién solicitado en ambas pantallas (de cocina/barra y camarero), siendo este el punto denotado con un “1” en el diagrama, con la siguiente apariencia:

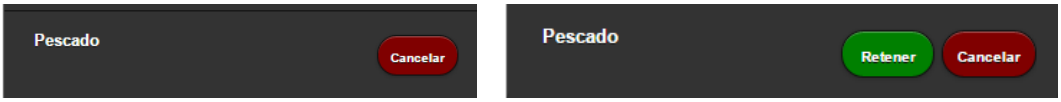

Figura 6-5-2. Estado inicial de los botones en camarero y cocina/barra

De esta manera, pasamos a describir la situación indicada en el punto “2” del diagrama 6-5-1, al producirse una retención del pedido, quedando este bloqueado para que no pueda ser cancelado, la apariencia cambiará nuevamente para denotar la nueva situación, en la que el camarero no podrá interactuar con el pedido, y el cocinero dispondrá del botón con el que confirmar que el pedido se ha completado o bien indicar que lo cancela por la razón que fuera:

Figura 6-5-3. Estado retenido de los botones en camarero y cocina/barra

Pasamos ahora al punto “3” del diagrama, donde el cocinero ha confirmado el pedido y el camarero debe pasar a recogerlo para llevarlo hasta la correspondiente mesa. En esta situación el cocinero ya no verá el estado de dicho pedido, ya que al haber sido completado, no supone una tarea pendiente que deba tener en pantalla:

Figura 6-5-4. Estado completado del botón en el perfil de camarero

Tras esto, en el punto “4” del diagrama, el pedido desaparecerá de la pantalla del camarero, al estar ya entregado en mesa, pasando dicho pedido a una lista

correspondiente a la mesa, indicando además el precio del producto para una eventual solicitud de cuenta en el perfil de caja:

Figura 6-5-5. Estado del pedido en caja, una vez se ha entregado

Por último, mostramos la apariencia de los botones en el caso de que un pedido hubiese sido cancelado por alguna de las dos partes:

Figura 6-5-6. Estado cancelado de los botones en camarero y cocina/barra

6.6 Consideraciones de diseño

Una vez descrito como hemos implementado el sistema para un restaurante, explicaremos algunas consideraciones de diseño que se han tenido que realizar, principalmente con la idea de conseguir la adaptabilidad a los diferentes tamaños de pantalla de los dispositivos en los que instalar la aplicación.

Para dicha adaptación será necesario el uso de las hojas de estilo CSS, con las que variaremos algunos parámetros de los elementos en pantalla, en función de las dimensiones de la misma. Esto deberemos hacerlo para que al cambiar el tamaño de pantalla de un dispositivo a otro, los botones y textos introducidos en la aplicación no se solapen llegando a situarse unos encima de otros, perdiendo por completo cualquier forma de interacción usuario-aplicación.

En primer lugar, lo que deberemos hacer será crear varias reglas CSS en orden jerárquico, de manera que sólo una sea utilizada en cada momento. Dichas reglas entrarán en acción cuando la pantalla tenga un ancho mínimo, y situaremos la más reestrictiva (la de menor ancho de pantalla) más arriba en el documento CSS, con lo que según las normas de prioridad de CSS [19], las sucesivas normas descritas debajo de esta tendrán más prioridad siempre que se cumplan las condiciones de aplicación de cada una.

De esta manera tendremos normas como las descritas en la siguiente figura:

```
@media all{
    APLICABLE SIEMPRE
}
@media all and (min-width: 30em){
    APLICABLE CON 480px O MÁS
}
@media all and (min-width: 35em){
    APLICABLE CON 560px O MÁS
}
@media all and (min-width: 42em){
    APLICABLE CON 672px O MÁS
}
```

Figura 6-6-1. Ejemplo de CSS utilizado para adaptación de tamaños

Con estas normas que se han utilizado, conseguiremos aplicar unos parámetros de estilo que necesitemos en cada caso, con el objetivo de que los elementos en pantalla se adapten al tamaño en cada momento.

El principal cambio que se realizará en cada “escalón” será el uso de un tamaño de letra menor para menores anchos de pantalla, haciendo de esta manera que los botones creados por la librería jQuery Mobile disminuyan rápidamente de tamaño y prácticamente no sea necesaria ninguna norma adicional.

Para los menores tamaños de pantalla se optó, además por disminuir los márgenes laterales de los botones, de forma que evitaramos el solapamiento de botones adyacentes en los grids usados.

Por último, se hizo necesario el uso de la regla de estilo *white-space: pre-wrap;*, con la que conseguiremos que un botón que tenga un texto considerablemente largo y que pudiera solaparse con otro adyacente debido a la longitud del texto, separe dicho texto en dos líneas, evitando el solapamiento.

Figura 6-6-2. Ejemplo de CSS con y sin la norma white-space: pre-wrap

Como vemos en la figura 6-6-2, en la primera imagen los dos botones están prácticamente pegados, muy cerca de producirse un solapamiento. Esto se evito con el uso de la norma ya comentada, obteniendo como resultado la separación de la imagen de la derecha.

CAPÍTULO 7. Pruebas y resultados

En este capítulo se realizarán las pertinentes pruebas finales para la comprobación del funcionamiento general del sistema implantado, así como la exposición de los resultados obtenidos en las mismas.

7.1 Preparación de las pruebas

Para la realización de las pruebas de funcionamiento del sistema desarrollado, en primer lugar necesitaremos empotrar nuestra web en los dispositivos móviles, para lo que utilizaremos PhoneGap, como ya se nombró en la introducción de esta memoria.

Como ya se comentó en el capítulo de tecnologías a utilizar, PhoneGap es una librería que permite la visualización de páginas web desarrolladas con HTML5, JavaScript y CSS en dispositivos móviles, permitiendo utilizar dichas páginas como aplicaciones móviles en los diversos sistemas operativos con la ventaja de reducir los costes de desarrollo al no tener que implementar una solución para cada sistema operativo.

Para la instalación de PhoneGap en nuestro ordenador y poder de esta manera generar un archivo instalable para nuestro dispositivo móvil, en este caso, Android, necesitaremos primero descargar el entorno de desarrollo Eclipse.[20]

Para esto, bastará con buscar en internet Eclipse ADT (Android Developer Tools) y descargarlo de forma gratuita [21]. Dicho entorno nos permitirá descargar de manera sencilla las librerías necesarias para desarrollar en Android, conocidas como SDK (Software Developers Kit), para el que deberemos descargar las últimas APIs de Google para el desarrollo en Android.

Tras esto, y si no dispusiéramos de un dispositivo Android donde realizar nuestras pruebas, podríamos utilizar el simulador integrado en la plataforma Eclipse, aunque en nuestro caso utilizaremos un dispositivo real para esto.

El siguiente paso que deberemos tomar, será instalar Node.js, una plataforma desarrollada en JavaScript que será la encargada de compilar nuestra aplicación [22]. Una vez instalada, procederemos a utilizar la consola de Node.js con permisos

de administrador, desde la que instalaremos Cordova CLI (Command Line Interface), la herramienta encargada de realizar el despliegue de la aplicación ya compilada en nuestro dispositivo.[23]

Para la instalación de Cordova bastará con utilizar la siguiente línea de comandos en la consola de Node.js nombrada anteriormente:

```
$ sudo npm install -g cordova
```

Tras esto, procederemos a crear un nuevo proyecto que nos servirá de base para empotrar nuestro código, lo que podremos hacer con los siguientes comandos:

```
$ cordova create restaurante com.proyecto.restaurante Restaurante

$ cd restaurante

$ cordova platform add android
```

Con esto se habrá creado nuestro proyecto en la nueva carpeta llamada “restaurante” en la que se habrán creado los archivos necesarios para poder desplegar una aplicación desarrollada en JavaScript en un dispositivo Android. De hecho, la creación de este proyecto dará como resultado un proyecto ejemplo ejecutable.

Navegando por las carpetas recientemente creadas, accederemos a los ficheros html y js de dicho proyecto ejemplo, con lo que simplemente sustituyendo estos por los nuestros, y recompilando el proyecto, conseguiremos que se genere el archivo instalable de nuestra aplicación. Dicha compilación se realizará con la línea de comandos:

```
$ cordova build
```

Y para lanzarlo a nuestro dispositivo Android conectado vía USB:

```
$ cordova run android
```

Obteniendo rápidamente nuestra aplicación en nuestro dispositivo móvil, adaptándose al tamaño de pantalla del mismo:

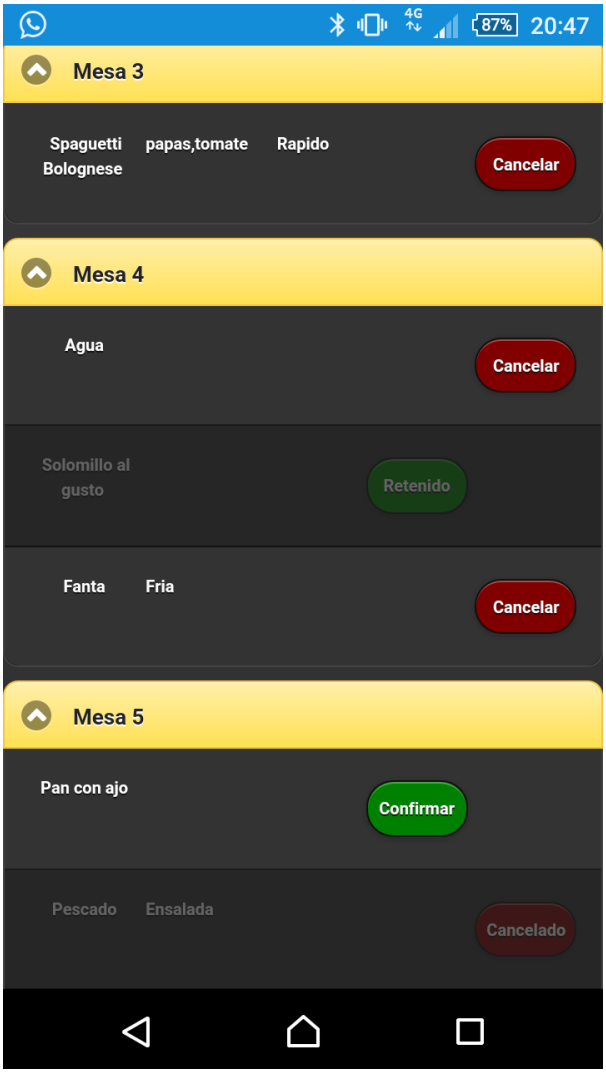

Figura 7-1-1. Apariencia de la aplicación desplegada en un dispositivo Android real

Para el despliegue en un teléfono iPhone necesitaremos un ordenador Apple, ya que es necesario usar el software Xcode para crear y desplegar el proyecto en un teléfono físico. Otra opción será utilizar el sistema operativo en nuestro ordenador con alguna de las herramientas existentes, como VirtualBox. Una vez tengamos esto, seguiremos los siguiente pasos:

- Instalar PhoneGap para iOS, desde la web oficial de PhoneGap
- Inicial un nuevo proyecto en Xcode (descargarlo de forma gratuita si no lo teníamos previamente). Este nuevo proyecto nos ofrecerá, ahora que hemos instalado el plugin de PhoneGap, una opción “Cordova-based application”.
- Una vez se haya creado el proyecto, le damos a ejecutar. Esto dará un error, pero nos creará la carpeta necesaria donde nosotros introduciremos nuestro código, análoga a la carpeta ‘www’ que generamos en Android.
- Tras esto, procederemos a introducir nuestro ficheros en dicha carpeta, con lo que conseguiremos que nuestro proyecto web ya sea desplegable en iOS y sólo nos quedará ejecutarlo para ver el resultado.

Figura 7-1-2. Apariencia de la aplicación desplegada en un dispositivo iOS real

Como vemos, hemos conseguido que la aplicación se adapte a la pantalla de un iPhone, como esperábamos, además de ver cómo se muestra el botón de atrás necesario para la navegación en iOS, ya que no disponen de un botón hardware.

Una vez desplegado el sistema en un dispositivo móvil, estaremos en disposición de realizar pruebas reales de rendimiento del sistema, utilizando un ordenador como sistema de cocina, que recibirá las peticiones del camarero. Dicho rol de camarero será llevado a cabo por otro usuario utilizando el terminal móvil en el que se acaba de instalar la aplicación.

Tras haber realizado estas pruebas en las que se han intercambiado peticiones entre camarero y cocinero durante unos 20 minutos, se ha comprobado el correcto funcionamiento del sistema, resultando en un retardo medio de refresco de unos 5 segundos. Recordemos que la pantalla se refresca cada 3 segundos, con lo que los 2 segundos restantes vienen dados por las peticiones al servidor de cada pedido, que dependerá en gran medida de las condiciones de la red instalada.

En el momento de las pruebas, se utilizó una red wifi pública, con unas velocidades de subida/bajada de 2Mbps/4Mbps, como se puede ver en la siguiente figura:

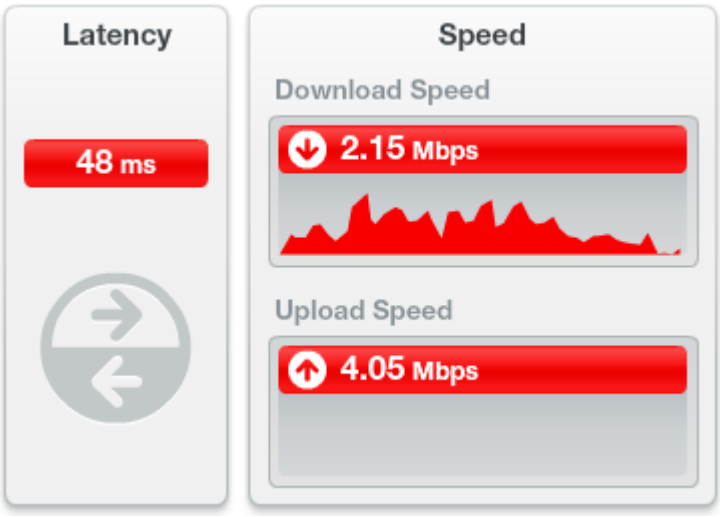

Figura 7-1-3. Condiciones de red en el momento de las pruebas[24]

Lo que, sin ser una red todo lo veloz que desearíamos, nos sirve para comprobar el rendimiento de nuestra aplicación en condiciones alejadas de la idealidad, obteniendo que la velocidad de refresco (además de los 3 segundos establecidos para realizar las peticiones) variaba desde los 170ms hasta los 20,15 segundos en el peor caso, situándose más del 90% de las peticiones por debajo de los 2 segundos.

Por último comentar que el consumo de datos realizados durante los 20 minutos que duró la sesión de pruebas se sitúa en 1.7MB, lo que extrapolado a una jornada de trabajo de 8 horas y 20 días laborales al mes, nos daría una estimación de 816MB consumidos de una eventual tarifa de datos móvil. Esto hace, debido a la cantidad de datos consumidos, que sea necesario el uso de una instalación wifi para el despliegue del sistema en un establecimiento, ya que de lo contrario estaríamos asumiendo un coste bastante alto de datos móviles de los terminales.

7.2 Batería de tests

Tras haber realizado pruebas con dispositivos reales, procederemos a implementar una batería de test con la que demostrar el funcionamiento general del sistema.

Para la realización de estas pruebas se ha incluido una batería de test con algunas de las funciones más significativas del sistema, como son la generación de una nueva categoría y un nuevo producto en el perfil de administrador, la generación y confirmación de un pedido, en camarero y cocina y el cierre de mesa una vez se ha cobrado.

Dicha batería de test se ha implementado con la librería de jQuery, *QUint*, que nos permite de una manera rápida y fácil, obtener un diagnóstico para las funciones a analizar.[25]

Un ejemplo de test podría ser el siguiente:

```
var objTest = new Objeto;
objTest.set("retenido", obj.attributes.retenido);

obj.unset("parent3");
obj.set("retenido",false);
obj.set("hecho",true);

query2.find({
   success: function (results2) {
      obj.set("parent3",results2[0]);

      // TEST
      obj.save({
         success: function(){
            QUnit.test( "Confirma preparación", function( assert ) {
               console.log(objTest.attributes.retenido+' - '+
                  obj.attributes.hecho+' - '+
                  obj.get("parent3").attributes.name);

               assert.ok( objTest.attributes.retenido == true &&
                  chosenRole == "cooker" &&
                  obj.attributes.hecho ==true &&
                  obj.get("parent3").attributes.name == "waiter", "Pedido preparado correctamente" );
            });
         }
      });
   },
   error: function(error) {
      alert("Error: " + error.code + " " + error.message);
   }
});
```

Figura 7-2-1. Ejemplo de código de test de QUnit

Donde, como se puede observar, denotado en rojo, se utiliza la función “test” de la librería *QUnit* para iniciar la comprobación de resultados, generando una salida visible en el documento HTML con la función *“assert”*, donde indicaremos las condiciones que deben darse para que el test sea considerado válido. En este caso, se trata del test que confirma que el pedido ha sido correctamente confirmado por el cocinero, ya que, como se puede ver en la figura 7-2-1, el rol de usuario en uso es “cooker” (cocinero), el flag “hecho” se ha puesto a “true”, y la cola a la que se dirige el pedido tras la confirmación (“parent3” del objeto Order) es la de camarero (“waiter”).

Una vez modificado el código para incluir las líneas necesarias de los test, procedemos a realizar las acciones a testear, en el orden siguiente:

- Creación de una nueva categoría en el perfil de administrador
- Creación de un nuevo plato en el perfil de administrador
- Creación de un nuevo pedido por parte del camarero
- Confirmación de preparación por parte del cocinero
- Solicitud de la cuenta por parte del camarero
- Confirmación de generación de cuenta en el perfil de caja
- Confirmación de entrega de cuenta en mesa, por parte del camarero
- Cierre de mesa en el perfil de caja, eliminándola para futuros clientes

Tras realizar estas acciones, comprobamos el resultado del test realizado. Resultado obtenido en la página principal de la aplicación al ser aquí donde se han situado los contenedores HTML que la librería *QUnit* utiliza para mostrar sus resultados, obteniendo:

Figura 7-2-2. Resultado de los test realizados (desplegable cerrado)

Como se puede ver en la figura 7-2-2, nuestro test ha diagnosticado que dos de nuestros test ha resultado fallido, en particular, los referentes a confirmación de generación de cuenta en el perfil de caja y confirmación de entrega de cuenta en mesa, por parte del camarero.

En este mismo apartado de resultados podremos desplegar dichos errores, con el objetivo de ver de qué se trata:

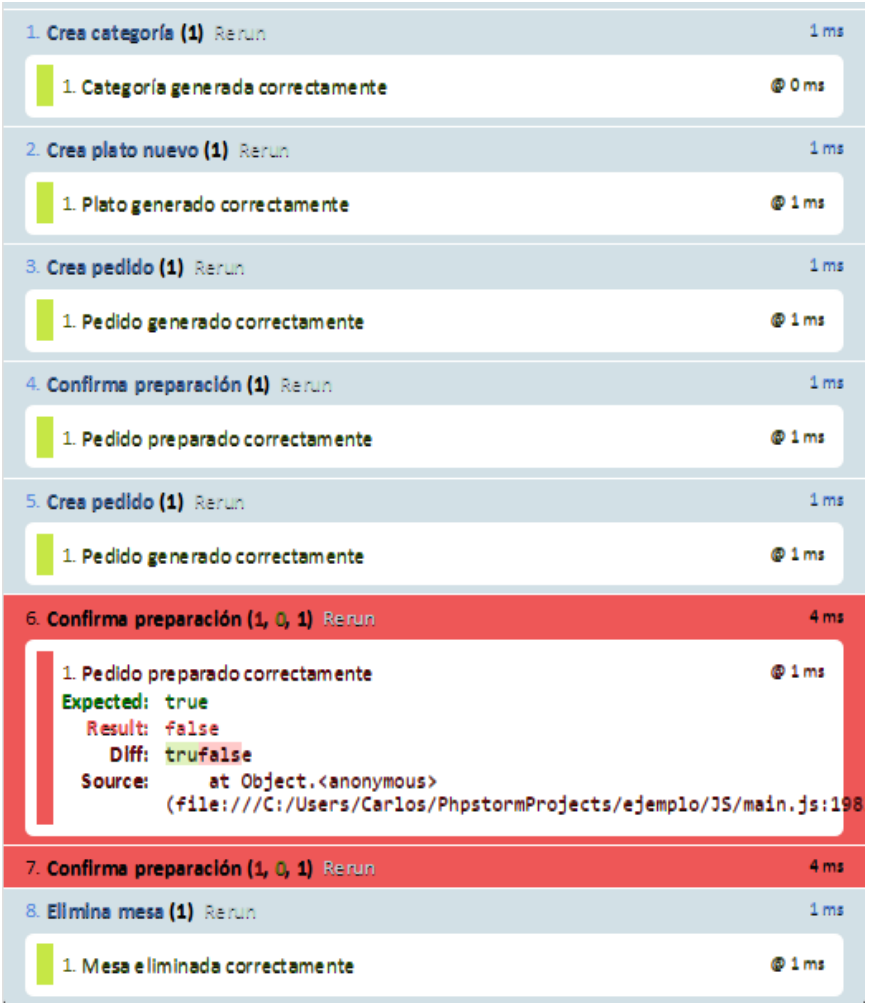

Figura 7-2-3. Resultado de los test realizados (desplegable abierto)

Donde, teniendo en cuenta las condiciones del test (figura 7-1-1), podemos asegurar que el test ha fallado, pero la función se ha realizado correctamente, ya que los dos errores detectados se han producido debido a que ese test en particular estaba diseñado para comprobar la confirmación de preparación de pedido por parte del cocinero, y nos devolvió error en situaciones diferentes a esa, como son la confirmación de generación de cuenta en el perfil de caja y la confirmación de recogida de cuenta por parte del camarero.

CAPÍTULO 8. Conclusiones y líneas futuras

En este capítulo se describen las conclusiones obtenidas del desarrollo de este Proyecto Fin de Carrera, así como las líneas de trabajo futuras.

8.1 Conclusiones

Una vez acabada la implementación y despliegue de nuestro sistema restaurante, podemos concluir que el flujo seguido para el desarrollo ha posibilitado que no tuviéramos que programar el código de la aplicación para cada uno de los sistemas operativos, ya que con el uso de PhoneGap nos permite empotrar una página web en cualquier dispositivo móvil.

Esto, además de habernos ahorrado el tiempo de implementación de cada una de las plataformas, nos ha permitido evitar la curva de desarrollo que supondría llegar a dominar cada uno de los lenguajes de programación en los que se implementan los diferentes sistemas operativos, como Swift en iOS, el sistema operativo de los iPhone, y Visual Basic o C# para Windows Phone.

Esta técnica, como hemos visto, también nos ha permitido esquivar el problema de los diferentes tamaños de pantallas en los móviles actuales, consiguiendo con el uso de normas CSS y la librería jQuery Mobile, que nuestra página web sea completamente adaptable.

Por otro lado, y gracias al hecho de que PhoneGap funciona con código JavaScript, y por extensión jQuery, permite que el aprendizaje sea bastante más rápido que el aprendizaje derivado de desarrollar para una plataforma móvil estándar, ya que se trata de uno de los lenguajes de programación actuales más utilizados en el ámbito del desarrollo web. Esto supone que encontremos con mucha facilidad referencias en Internet que puedan ayudarnos a dar los primeros pasos en el uso del lenguaje.

Sin embargo, uno de los puntos en contra que puede tener el uso de PhoneGap para el desarrollo de aplicaciones móviles es precisamente una de sus ventajas, como es el hecho de que el diseño de las páginas web sea estándar para todas las plataformas. Esto, que a priori podría parecer una ventaja, se vuelve un pequeño inconveniente cuando entramos en el campo de la usabilidad de usuario.

Es en este apartado donde encontramos el mayor inconveniente del uso de PhoneGap, ya que los usuarios de un determinado sistema operativo se acostumbran al uso de aplicaciones nativas para esa plataforma, y al diseño de las aplicaciones adaptado a dichos sistemas operativos. Para explicar esto un poco más, recurriremos a la siguiente figura, en la que vemos el diseño de una misma aplicación en dos plataformas móviles diferentes, en este caso iOS y Android.

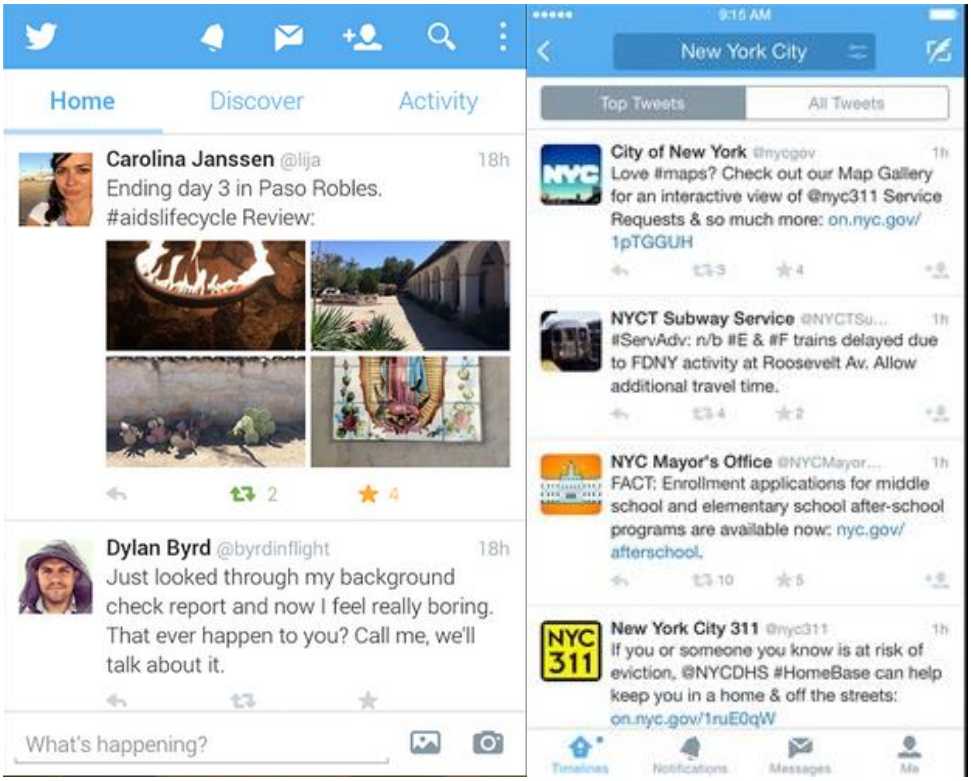

Figura 8-1-1. Aplicación móvil en dos sistemas operativos diferentes(Android / iOS)

Como se puede ver en las imágenes, a la izquierda, la aplicación de Twitter para Android, a la derecha, Twitter para iOS, el diseño, aunque similar en la forma que se muestran los twitts, es muy diferente en las barras de navegación superior e inferior, en la que Android utiliza botones para acceder a las diferentes secciones de la aplicación, utilizando el botón hardware de Android para volver a pantallas anteriores. iOS, por su parte, sitúa los botones para acceder a las diferentes secciones en la barra de navegación inferior. Además, para la navegación utiliza un botón de retroceso, en la parte superior izquierda.

A esto hay que añadirle el apartado de escritura de nuevos twitts, para lo que Android sitúa una línea de texto en la parte inferior con la que el usuario puede rápidamente empezar a redactar una publicación. iOS, por su parte, reserva un botón en la parte derecha de la barra de navegación superior para esto.

Como vemos, el diseño nativo de las aplicaciones en diferentes dispositivos, dista mucho de ser igual, con lo que el uso de PhoneGap, al implementar un mismo interfaz de usuario, podría hacer que los usuarios se sintieran un poco perdidos en el uso de la aplicación.

Una forma de mitigar esto podría ser el uso de algunos plugins existentes en JavaScript que nos permiten de forma sencilla, reconocer el navegador donde se está ejecutando una determinada aplicación. A partir de esta información extraída podríamos hacer pequeños cambios en los interfaces de usuario de nuestra aplicación, ya que el navegador predeterminado en iOS es el Safari, por lo que si detectásemos esto, podríamos personalizar la aplicación para un iPhone.

Uno de estos posibles cambios podría ser, por ejemplo, ocultar un determinado botón en alguno de los navegadores, como hemos realizado en nuestra aplicación, en la que “escondemos” el botón de “atrás” en todos los navegadores que no sean Safari, con lo que los dispositivos Android y Windows Phone podrán navegar con el botón de atrás físico que disponen los terminales, ayudando así a que en usuario no note un cambio en la forma de interactuar con la aplicación en este sentido.

El uso de dichos plugins, aunque podría ayudarnos, como ha hecho en nuestra plataforma, tampoco nos libraría del todo del problema, ya que un cambio de diseño en las pantallas de la aplicación con el objetivo de adaptarnos para intentar ser más “nativo” implicaría un aumento en el tiempo de diseño e implementación de la aplicación, perdiendo así gran parte de la ventaja competitiva que ofrece PhoneGap al reducir el TTM (*time to market*).

Por otro lado, y centrándonos en el punto de vista del empresario, implementando el sistema de esta manera conseguiremos que, con una inversión mínima, se pueda obtener un sistema que permite la interacción total entre los trabajadores dentro del restaurante, optimizando su trabajo al máximo. Dicha inversión implicaría únicamente los costes de adquisición del sistema y el pago de una suscripción a Internet, dado que la aplicación podría ser desplegada en los teléfonos móviles personales de los trabajadores, liberando de estos costes al propietario del restaurante.

Es esta una de las principales razones que darían una ventaja competitiva a nuestro sistema frente a otros sistemas ya implementados en el mercado, como la aplicación utilizada en la cadena de restaurantes Foster's Hollywood, en la que los camareros realizan los pedidos de forma telemática desde un dispositivo móvil, de la misma forma que lo hace nuestro sistema. La diferencia radica en que los terminales utilizados en dichos restaurantes son iPods pertenecientes a la propia empresa, con los costes de adquisición que esto habrá conllevado a los propietarios del restaurante.

De esta manera, y para concluir, podríamos afirmar que con este trabajo se han fijado las bases y conocimientos para llevar a cabo de forma exitosa este tipo de diseño de aplicaciones multiplataforma, fijando una serie de normas y técnicas a seguir para lograr un desarrollo ágil y sin altos costes de implementación.

Se ha comprobado también que PhoneGap es una potente herramienta con la que podemos crear aplicaciones multidispositivo de forma rápida y estandarizada, pero que implicará una pérdida en la experiencia de usuario que podría influir directamente en una caída de ventas de la aplicación.

Por otro lado, podemos afirmar también que PhoneGap puede ser una opción ideal con la que desarrollar una aplicación en la que el diseño no sea primordial,

como es el caso de nuestro sistema de restaurante, donde primará más la eficiencia que un interfaz atractivo hacia el usuario.

8.2 Líneas Futuras

Con el desarrollo de este Proyecto Fin de Carrera hemos visto como podemos realizar aplicaciones móviles multidispositivo de forma sencilla, con el uso de JavaScript y PhoneGap. Sin embargo, surgen, como ya se ha comentado en el apartado de conclusiones, algunas posibles mejoras implementables a nuestro sistema.

En primer lugar, a nivel de diseño podría ser beneficiosa la inclusión de algún tipo de personalización centrada en el restaurante en el que desplegar el sistema. Esto dotaría a la aplicación de una personalidad única y centrada en el establecimiento, lo que la haría, además, mucho más atractiva para el empresario responsable del restaurante, pudiendo este tener mayor disposición a realizar un desembolso mayor en una eventual compra del sistema.

Por otro lado, podría ser beneficioso el uso de una pantalla de “login”, en la que cada usuario accedería a su pantalla principal tras iniciar su sesión de usuario con unas credenciales previamente registradas en el sistema. De esta forma podríamos, además, aumentar la seguridad del sistema o posibilitar algún tipo de personalización de cada usuario, como cambiar el tema de la aplicación.

Además de esto, otra mejora importante a introducir en el sistema, sería la inclusión de un perfil de usuario donde los clientes del restaurante pudieran ver la carta en algún móvil o Tablet, incluyendo fotos de los platos de la carta disponibles en cada momento, generando de esta manera un valor añadido al restaurante.

Al hilo de esto, y basándonos en la idea comentada en el párrafo anterior, podríamos hacer que los usuarios del restaurante, mediante la aplicación pudiese realizar los pedidos directamente a la cocina, sin la necesidad de la mediación del camarero, que, al no tener que tomar notas de las mesas entrantes, podría aprovechar ese tiempo en mejorar la velocidad del servicio en la entrega de pedidos de la cocina a las mesas.

Esta misma idea, reportaría al cliente una mayor satisfacción al poder ver en tiempo real el estado de su pedido, pudiendo observar, por ejemplo, si su plato solicitado ha entrado ya a preparación (ha sido retenido en cocina) o está todavía en cola.

Por último y como mejora adicional, se plantea la opción de que la base de datos almacenara algún tipo de registro de los platos solicitados, rechazados, etc. Esto crearía un valor añadido para el empresario, que podría utilizar esta información para mejorar el servicio ofrecido, ya que podríamos ofrecerle una pantalla con estadísticas del restaurante como el tiempo medio de preparación de pedidos o el número de pedidos realizados de cada plato. De esta manera podríamos, incluso, ayudar al encargado del restaurante a llevar un inventario detallado, al quedar registrado todo lo que se ha consumido en el establecimiento.

Bibliografía

[1]: http://appleweblog.com/2008/11/nokia-pierd-cuota-de-mercado-frente-a-apple-y-rim
Noticia sobre la caída en cuota de mercado de Nokia ante Apple en 2008, cuando se empezaba a comercializar smartphones. [Fecha de consulta: 09/09/13. Enlace revisado: 18/06/15]

[2]: http://www.tuexperto.com/2007/11/28/nokia-se-reafirma-samsung-asciende-y-motorola-cae-en-el-mercado-mundial-de-moviles/
Noticia en la que se indica la cuota de mercado de Samsung, actual líder del mercado, antes del 'boom' de los teléfonos inteligentes. [Fecha de consulta: 09/09/13. Enlace revisado: 18/06/15]

[3]: http://www.xatakamovil.com/mercado/los-smartphones-ya-superan-en-ventas-mundialmente-a-los-telefonos-moviles-convencionales
Noticia en la que se señala que la venta de teléfonos inteligentes supera las ventas de teléfonos convencionales. [Fecha de consulta: 10/09/13. Enlace revisado: 18/06/15]

[4]: http://www.spainmovil.es/economia-y-mercado/noticias/n25234/pantallas-pulgadas.html
Noticia de donde se tomó la imagen para ilustrar la variedad de tamaños de pantalla con las que los desarrolladores de aplicaciones tienen que lidiar [Fecha de consulta: 11/09/13. Enlace revisado: 18/06/15]

[5]: http://es.wikipedia.org/wiki/HTML
Información general sobre HTML. [Fecha de consulta: 20/05/15. Enlace revisado: 18/06/15]

[6]: http://es.wikipedia.org/wiki/HTML5#Diferencias_entre_HTML5_y_HTML4.2FXHTML
Diferencias entre los estándares HTML4 y HTML5. [Fecha de consulta: 20/05/15. Enlace revisado: 18/06/15]

[7]: http://es.wikipedia.org/wiki/JavaScript
Información general sobre JavaScript. [Fecha de consulta: 20/05/15. Enlace revisado: 18/06/15]

[8]: http://es.wikipedia.org/wiki/AJAX
Información general sobre AJAX en JavaScript. [Fecha de consulta: 20/05/13. Enlace revisado: 18/06/15]

[9]: Brad Broulik, 2011, Pro jQuery Mobile
Libro introductorio sobre jQuery, su uso y sus ventajas de uso en el desarrollo de webs adaptables a dispositivos móviles.

[10]: http://es.wikipedia.org/wiki/JQuery
Web de consulta introductoria a jQuery. [Fecha de consulta: 20/05/13. Enlace revisado: 18/06/15]

[11]: http://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada
Web de consulta introductoria a CSS. [Fecha de consulta: 20/05/13. Enlace revisado: 18/06/15]

[12]: http://parse.com/products

Web de Parse donde se hace una introducción de lo que es Parse, y cómo funciona. [Fecha de consulta: 20/05/13. Enlace revisado: 18/06/15]

[13]: Thomas Myer, 2012, Beginning PhoneGap
Libro de introducción a PhoneGap.

[14]: https://www.parse.com/docs/js/api/symbols/Parse.html
Documentación de la API de Parse, donde podemos encontrar las funciones específicas para conectarnos a la base de datos, además de los métodos con los que hacer peticiones a dicha base de datos. [Fecha de consulta: 12/10/13. Enlace revisado: 18/06/15]

[15]: https://jquerymobile.com/
Web de jQuery Mobile, donde se puede encontrar una descripción detallada de la librería y la documentación necesaria para su uso. [Fecha de consulta: 03/10/13. Enlace revisado: 18/06/15]

[16]: http://api.jquerymobile.com/data-attribute/
API de jQuery Mobile donde podemos comprobar los atributos HTML que podemos incluir en los elementos de nuestro DOM de manera que la librería los personalice. [Fecha de consulta: 03/10/13. Enlace revisado: 18/06/15]

[17]: http://view.jquerymobile.com/1.3.2/dist/demos/widgets/collapsibles/
Documentación de jQuery Mobile donde podemos ver las diferentes opciones disponibles para personalizar nuestros menús desplegables, así como ejemplos de código que incluir en nuestro sistema. [Fecha de consulta: 03/10/13. Enlace revisado: 18/06/15]

[18]: http://demos.jquerymobile.com/1.2.0/docs/pages/dialog/index.html
Documentación de los cuadros de los menús emergentes de jQuery Mobile donde podemos ver las diferentes personalizaciones que ofrece. [Fecha de consulta: 10/11/13. Enlace revisado: 18/06/15]

[19]: https://librosweb.es/libro/css/capitulo_13/prioridad_de_las_declaraciones_css.html
Documento donde se habla del orden de prioridad de las normas en CSS. [Fecha de consulta: 22/06/15. Enlace revisado: 22/06/15]

[20]: https://eclipse.org/
Página principal de Eclipse, donde podemos encontrar el enlace de descarga del IDE. [Fecha de consulta: 28/12/14. Enlace revisado: 18/06/15]

[21]: http://developer.android.com/tools/sdk/eclipse-adt.html
Página de desarrolladores de Android donde encontraremos instrucciones y enlaces de descarga de los kits de desarrollo de Android. [Fecha de consulta: 28/12/14. Enlace revisado: 18/06/15]

[22]: https://nodejs.org
Página principal de la plataforma NodeJS, donde encontraremos documentación y enlaces de descarga. [Fecha de consulta: 28/12/14. Enlace revisado: 18/06/15]

[23]: http://www.pubnub.com/blog/how-to-convert-your-javascript-app-into-an-android-app-with-phonegap/

Tutorial de instalación de PhoneGap y despliegue en un dispositivo Android. [Fecha de consulta: 03/06/15. Enlace revisado: 18/06/15]

[24]: http://www.adsl4ever.com/test/11/
Página donde podemos realizar un test de velocidad a la red a la que estemos conectados en ese momento. [Fecha de consulta: 16/06/15. Enlace revisado: 18/06/15]

[25]: https://qunitjs.com/
Página principal de QUnit en la que, además de encontrar una pequeña descripción inicial, encontramos un ejemplo inicial de uso. [Fecha de consulta: 13/06/15. Enlace revisado: 18/06/15]

[26]: Página web del Colegio Oficial de Ingenieros de Telecomunicación, www.coit.es.
Consultada en Junio de 2015

Pliego de Condiciones

Esta parte del documento se dedica íntegramente a enumerar los recursos utilizados para el desarrollo del presente Proyecto Fin de Carrera, tanto software como hardware.

Pliego de Condiciones

Los recursos hardware empleados en el desarrollo del presente proyecto son los siguientes:

- **Ordenador portátil Lenovo G510**: En él se ha llevado a cabo la implementación de la totalidad del código del sistema desarrollado, así como la realización de esta memoria. Las características principales de este PC son:

    - Procesador Inter Core i7
    - 8 GB de memoria RAM
    - 1 TB de disco duro
    - 2GB de memoria gráfica

- **Teléfono móvil Sony Xperia Z3 Compact:** En él se han llevado a cabo las pruebas de la aplicación en el sistema operativo Android.

    - Procesador Quad-core a 2,5GHz
    - 3 GB de memoria RAM
    - 16 GB de memoria

Por otro lado, los recursos software empleados para desarrollar los objetivos del presente proyecto son los siguientes:

- **PhpStorm 7.1:** Programa empleado para la escritura y depuración del código escrito tanto en HTML como en jQuery y CSS.

- **Microsoft Office 2013:** Suite ofimática utilizada para la redacción de la presente memoria del Proyecto Fin de Carrera.

- **Google Chrome:** Navegador web utilizado para el despliegue y depuración de la web desarrollada, principalmente, con el inspector de elementos del navegador.

Presupuesto

En este apartado se presenta detallado el presupuesto calculado para la realización de este Proyecto Fin de Carrera. En él aparece la tarificación del trabajo realizado por el Ingeniero de Telecomunicación pertinente, los gastos de material, con su respectiva amortización, y los gastos ocasionados por la redacción del proyecto. Además, en la última parte de este capítulo puede verse el presupuesto global.

Presupuesto

P.1 Trabajo tarificado por tiempo empleado

Se contabilizan los gastos que corresponden a la mano de obra, según el salario correspondiente a la hora de trabajo de un ingeniero. El COIT propone utilizar la siguiente fórmula:

$$H = (C \times 74{,}88 \times H_n) + (C \times 96{,}72 \times H_e)\ €$$

donde:

- H son los honorarios totales por el tiempo dedicado.
- C es el coeficiente de corrección en función del número de horas trabajadas definido por el COIT por tramos.
- H_n son las horas normales trabajadas dentro de la jornada laboral.
- H_e son las horas especiales trabajadas.

Se estima que para la realización del presente proyecto se ha trabajado durante 9 meses, desde Octubre de 2013 hasta Abril de 2014. Considerando 20 días laborables al mes, con una dedicación a tiempo completo de 8 horas por jornada laboral, el número de horas normales asciende a:

$$H_n = 9 \times 20 \times 8 = 1440 \text{ horas}$$

Dado que no se ha trabajado fuera del horario laboral, el número de horas especiales es cero. Por otro lado, el coeficiente corrector a aplicar es el que está asignado por el COIT para un exceso de 1080 horas, cuyo valor es 0,4. Por tanto, finalmente se obtiene que el coste total de honorarios asciende a:

$$\boldsymbol{H} = (0{,}4 \times 74.88 \times 1440) + (0{,}4 \times 96{,}72 \times 0) = \mathbf{43.130{,}88\ €}$$

El trabajo tarifado por tiempo empleado asciende a la cantidad de *cuarenta y tres mil ciento treinta euros con ochenta y ocho céntimos*.

P.2 Amortización del inmovilizado material

En el inmovilizado material se consideran tanto los recursos hardware como software empleados para la realización de este Proyecto Fin de Carrera.

Se estipula el coste de amortización para un periodo medio de 5 años utilizando un sistema de amortización lineal en el que se supone que el inmovilizado material se desprecia de forma constante a lo largo de su vida útil. La cuota de amortización anual se calcula haciendo uso de la siguiente fórmula:

$$\text{Cuota Anual} = \frac{\text{Valor de adquisición - Valor residual}}{\text{Número de años de vida útil}}$$

donde el valor residual es el valor teórico que se supone que tendrá el elemento en cuestión después de su vida útil.

P.2.1 Amortización del material hardware

Dado que la duración de este Proyecto Fin de Carrera es de 9 meses y es inferior al periodo de 5 años estipulado para el coste de amortización, los costes serán derivados del tiempo de utilización de cada recurso.

En la Tabla P.1 se muestra el hardware necesario para la realización del proyecto, indicando para cada elemento su valor de adquisición, valor residual y coste de amortización. Los recursos hardware se han utilizado durante el período especificado en la columna “tiempo de uso” teniendo en cuenta que aproximadamente se han dedicado dos meses a cada módulo y tres meses al sistema integrador.

Elemento	Tiempo de uso	Valor de adquisición	Valor residual	Coste de la amortización
Ordenador Lenovo G510	9 meses	600,00 €	150,00 €	67,50 €
Teléfono Sony Xperia Z3 Compact	9 meses	500,00 €	60,00 €	126,00 €
Total		**1100,00 €**	**210,00 €**	**193,50 €**

Tabla P.1. Precios y costes de amortización del hardware.

El coste del material hardware asciende a *ciento noventa y tres euros con cincuenta céntimos.*

P.2.2 Amortización del material software

Para el cálculo de los costes de amortización del material software se considerará, al igual que con el material hardware, que los costes serán los derivados del tiempo de utilización de cada recurso.

La Tabla P.2 muestra los elementos software necesarios para la realización del proyecto, así como su valor de adquisición, valor residual y coste de amortización.

Elemento	Tiempo de uso	Valor de adquisición	Valor residual	Coste de la amortización
PhpStorm 7.1	9 meses	0,00 €	0,00 €	0,00 €
Microsoft Office 2013	9 meses	450,00 €	50,00 €	60,00 €
Google Chrome	9 meses	0,00 €	0,00 €	0,00 €
Total		**450,00 €**	**50,00 €**	**60,00 €**

Tabla P.2. Precios y costes de amortización del hardware.

Por tanto, el coste total del material software asciende a *sesenta euros.*

P.3 Redacción del proyecto

El COIT recomienda utilizar la fórmula siguiente para determinar el coste asociado a la redacción de la memoria del proyecto.

$$R = 0{,}07 \times P \times C$$

donde:

- *R* son los honorarios por la redacción del proyecto.
- *P* es el presupuesto.
- *C* es el coeficiente reductor de honorarios en función del presupuesto.

El valor del presupuesto *P* se calcula sumando los costes de las secciones anteriores correspondientes al trabajo tarifado por tiempo empleado y a la amortización del inmovilizado material, tanto hardware como software. Esta suma de los costes se muestra en la Tabla P.3.

Concepto	Coste
Trabajo tarifado por tiempo empleado	43.130,88 €
Amortización del material hardware	193,50 €
Amortización del material software	60,00 €
Total (*P*)	**43.384,38€**

Tabla P.3. Presupuesto incluyendo el trabajo tarificado y la amortización del material

El coeficiente corrector a aplicar, según el establecido por el COIT para el tramo comprendido entre 30,050 € y 60,131 €, es de 0,9. Por tanto, se obtiene:

$$\boldsymbol{R} = 0{,}07 \times 43.384{,}38 \times 0{,}9 = \mathbf{2.733{,}21\ €}$$

El coste de la redacción del proyecto, por tanto, asciende a *dos mil setecientos treinta y tres euros con veintiun céntimos*.

P.4 Derechos del visado del COIT

El COIT establece que para la redacción de proyectos y trabajos en general, los derechos de visado para 2009 se calculan en base a la siguiente fórmula:

$$V = 0{,}006 \times P \times C$$

donde:

- *V* es el coste de visado del proyecto.
- *P* es el presupuesto.
- *C* es el coeficiente reductor en función del presupuesto.

El valor del presupuesto *P* se halla sumando los costes de las secciones anteriores correspondientes al trabajo tarifado por tiempo empleado, a la amortización del

inmovilizado material, tanto hardware como software, y a la redacción del proyecto. Esta suma se muestra en la

Tabla P.4.

Concepto	Coste
Trabajo tarifado por tiempo empleado	43.130,88 €
Amortización del material hardware	193,50 €
Amortización del material software	60,00 €
Redacción del proyecto	2.733,21 €
Total (*P*)	**46.117,59 €**

Tabla P.4. Presupuesto incluyendo el trabajo tarifado, la amortización y la redacción

En este caso, el coeficiente reductor *C* es el asignado para el tramo entre 42.070,70 € y 63.106,05 €, cuyo valor es de 0,45. Así, aplicando este dato en la fórmula anterior, con los datos obtenidos en la

Tabla P.4 y el coeficiente especificado, se obtiene:

$$\boldsymbol{V} = 0{,}006 \times 46.117{,}59 \times 0{,}45 = \mathbf{124{,}51\ €}$$

Los costes por derechos de visado del presupuesto ascienden a *ciento veinticuatro euros con cincuenta y un céntimos*.

P.5 Gastos de tramitación y envío

Los gastos de tramitación y envío están estipulados en *nueve euros* (**9,00 €**).

P.6 Aplicación de impuestos y coste total

La realización del presente Proyecto Fin de Carrera está gravada por el Impuesto General Indirecto Canario, I.G.I.C., en un siete por ciento (7 %). En la Tabla P.5 se muestra el presupuesto final con los impuestos aplicados.

Concepto	Coste
Trabajo tarifado por tiempo empleado	43.130,88 €
Amortización del material hardware	193,50 €
Amortización del material software	60,00 €
Redacción del proyecto	2.733,21 €
Derechos de visado del COIT	124,51€
Gatos de tramitación y envío	9,00 €

Total (Sin IGIC)	**46.251,10 €**
IGIC (7%)	3.237,57 €

Total	**49.488,67 €**

Tabla P.5. Presupuesto total del Proyecto Fin de Carrera.

El presupuesto total del proyecto *"Técnicas para desarrollo de aplicaciones multidispositivo para empresas e implementación de caso de uso para un restaurante"* asciende a *cuarenta y nueve mil cuatrocientos ochenta y ocho euros con sesenta y siete céntimos*.

El ingeniero proyectista

Fdo: D. Carlos Pérez Pérez

En Las Palmas de Gran Canaria a 21 de Junio de 2015