ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Desarrollo Software para la Gestión de un Terminal Punto de Venta

***AUTOR*: Marcelo José Gómez Conesa**
***DIRECTORA*: Mª Francisca Rosique Contreras**
Abril, 2013

Autor	Marcelo José Gómez Conesa
E-mail del Autor	Marce89.jgc@gmail.com
Director(es)	Mª Francisca Rosique Contreras
Título del PFC	Desarrollo Software para la Gestión de un Terminal Punto de Venta
Descriptores	

Resúmen

En este proyecto se va a desarrollar una parte del software TPV a medida para una empresa del sector de la hostería.
La necesidad de dicho software surge de la necesidad de mejorar, modernizar y completar la funcionalidad del software que hasta ahora posee la empresa.
La parte a desarrollar en este proyecto consiste en el diseño e implementación tanto de la base de datos como de una aplicación para proveer de una interfaz gráfica a los empleados. Dicha aplicación gráfica debe servir para permitir gestionar las ventas de los empleados.
Así mismo, se añadirá un sistema de replicación de bases de datos, con el fin de mantener varios equipos en los establecimientos del cliente, los cuales mantendrán los datos actualizados en tiempo real sobre las ventas recogidas en todos los terminales. Con ello, se aporta tanto comodidad como una mayor disponibilidad de los datos.
Para la implementación de la aplicación se ha escogido el modelo de programación orientado a objetos utilizando el lenguaje de programación JAVA.
Como Sistema Gestor de Base de Datos se ha empleado PostgreSQL con una base de datos relacional para el almacenamiento masivo de pedidos y servicios.
Se ofrece también control horario y de rendimiento de los empleados mediante el registro de estadísticas tales como hora de entrada y salida, o ventas realizadas.

Titulación	Ingeniero Técnico en Telecomunicaciones, esp. Telemática
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	

INDICE

Índice de figuras

1. Introducción

Con el nivel de penetración que tiene la informática en todos los ámbitos de la sociedad actual, tanto a nivel de consumo como a nivel profesional, resulta lógico pensar en el aprovechamiento que las empresas obtienen y los beneficios que su explotación les supone.
En el mundo de la hostelería y restauración, como no podía ser de otra manera, la informática, mediante los sistemas TPV (Terminal Punto de Venta) o POS (de las siglas en inglés *Point of Sale*), ha supuesto una importante mejora en la calidad de servido y una reducción en los tiempos de espera de los clientes.
El empleo de un sistema TPV en este tipo de establecimientos es siempre recomendable, pero más aún si cabe cuando se trata de locales donde existe un alto índice de comensales por hora. La centralización y simplificación de la gestión puede evitar errores por interpretación de notas manuscritas u olvidos en caso de tomar notas verbales, por ejemplo. Además, la posibilidad de obtener todo tipo de estadísticas en tiempo real y de bloquear productos agotados instantáneamente permite modificar, en cualquier momento, el desempeño normal de las tareas rutinarias para adaptarse a la situación actual de una manera más precisa.

Sin embargo, a menudo, el software utilizado en dichos TPVs es un software comercial, genérico y con escasa adaptabilidad, lo que obliga a las empresas a adaptarse a las características que este ofrezca. Y no al contrario, que sería lo deseable. Por lo tanto, aunque resulta más caro, en muchas ocasiones es necesario recurrir a software "a medida". Este software es creado a partir de los requisitos específicos de la empresa que lo contrata. Por lo tanto, es un software que se ajustará a la perfección al modelo de negocio de la empresa y reportará un beneficio aún más potenciado que el simple uso de un TPV comercial.

Es precisamente esa necesidad la que impulsa el desarrollo de este proyecto. La necesidad de mejorar el sistema TPV de una empresa hostelera de la localidad murciana de Puerto de Mazarrón, llamada Helados Artesanos Venecia.
Dicha mejora del sistema TPV se ha dividido en tres partes:

- Aplicación para la gestión de las ventas.
- Aplicación para la gestión de los datos.
- Aplicación móvil para la gestión de las ventas.

Este proyecto consistirá en desarrollar la primera parte. En esta primera parte se diseñará la base de datos y se desarrollará interfaz gráfica de usuario que servirá para crear, editar y finalizar las ventas. Esta aplicación será utilizada por los empleados a modo de caja registradora.
La segunda parte consistirá en una aplicación para gestionar la base de datos, permitiendo añadir, retirar o modificar productos, empleados o pedidos, así como obtener estadísticas de productos vendidos, ventas de empleados, etc…
Y la tercera parte consiste en una aplicación para plataformas móviles que permita a los trabajadores portar PDA's con las que gestionar los pedidos de los clientes y enviarlos a cocina de forma telemática, reduciendo con ello de una manera notable el tiempo de espera de los clientes y mejorando, por tanto, la calidad del servicio a los cliente.

Por tanto, la primera aplicación, que se desarrollara en este proyecto, deberá cumplir una serie de requisitos u objetivos, proveyendo de capacidades que, si bien, no serán incorporadas en esta primera aplicación, permitirán la integración de los servicios posteriormente.

1.1. Objetivos

Como se ha dicho anteriormente, la aplicación debe cumplir ciertos objetivos funcionales y de soporte para la integración de futuros servicios.
El elemento más básico que debe diseñarse de forma óptima, pues será sobre lo que se asentará el conjunto de aplicaciones, es la base de datos.
El diseño, debe contemplar la escalabilidad a la que estará sometida, acumulando información de los pedidos de las diversas temporadas de trabajo.
Se debe dotar a los empleados con una interfaz gráfica con la que poder manipular los datos de la base de datos y las ventas. Dicha interfaz gráfica debe ser intuitiva a la par que eficiente, para reducir al máximo el tiempo de aprendizaje de los empleados y el tiempo de servicio. Una buena distribución dentro de la interfaz gráfica de usuario puede reducir notablemente el tiempo necesario para generar los pedidos o modificarlos.

A su vez, se debe controlar el hardware necesario para imprimir tiques de las cuentas de los clientes, tiques de los pedidos que enviarán de forma telemática los trabajadores (algo que se utilizará en la tercera parte del proyecto completo, según se ha explicado anteriormente) y hardware específico del cajón de dinero.
Los tiques de cliente deben tener una estructura específica y construirse con la información de los pedidos, indicando los productos, las cantidades de producto, el precio y el importe total de la cuenta.
Los tiques de pedidos telemáticos reflejarán los productos y cantidades, e incluirán modificaciones de productos tales como sabores, ingredientes extras, o cambios varios. Se utilizará una impresora diferente para tiques de clientes y productos.
El cajón de dinero se maneja a través de la impresora a la que se encuentre conectado, por medio de un comando que se envía a la misma. Dicho cajón deberá abrirse automáticamente cuando se cobre alguna cuenta.

Para garantizar la disponibilidad de los datos, en cada establecimiento habrán dos equipos realizando las funciones de TPV. Ambos equipos contarán con una base de datos cada uno, y para mantener la misma información en ambos se empleará un sistema de replicación de bases de datos. De esta manera, aunque uno de los equipos deje de funcionar, o se colapse, siempre estará garantizada la disponibilidad de la información sobre los pedidos tanto de terraza como de barra del local.
A su vez, el equipo que dejase de funcionar o que simplemente no se encontrase encendido, deberá actualizar su base de datos antes de ofrecer servicio a los trabajadores, para garantizar la integridad de los datos.

El hecho de tener varios equipos y en el futuro incorporar las PDA’s, hace que sea necesario realizar el diseño teniendo en cuenta los posibles problemas de consistencia que se pudieran dar debido al acceso a la información del mismo pedido por varios usuarios. Por lo tanto, se hace necesario contemplar algún sistema de exclusión mutua para garantizar también la consistencia de los datos.

Como toda empresa se rige por una estructura jerárquica en la que el puesto más alto lo ocupa el jefe, seguido por encargados o gerentes y finalmente los empleados de menor rango, es necesario mantener cierta funcionalidad supeditada a los diferentes permisos que pueda tener cada usuario. Por lo tanto, se deberá implementar algún sistema a tal fin.
Como se ha comentado anteriormente, el desarrollo de la aplicación incorporará funcionalidad adicional para la futura incorporación de nuevos servicios. En el caso de la incorporación de las PDA's, se dejará el sistema preparado para recibir peticiones de uso de las impresoras de forma remota.
En cuanto a la futura aplicación de gestión administrativa, se dejará la base de datos diseñada teniendo en cuenta requisitos o necesidades que habrá de cumplir para obtener estadísticas de productos vendidos (con el fin de mejorar la carta de productos en base al número de ventas que obtiene cada producto), de la venta por empleado (para llevar un control sobre el trabajo que realiza cada uno) así como de las cifras de ingresos totales de la empresa en función de las jornadas de trabajo.

1.2. Antecedentes.

Como se ha comentado, el objetivo del proyecto general es ofrecer una solución que mejore el software actual que el cliente posee (además de los objetivos didácticos). Por lo tanto, resulta necesario comentar ciertos aspectos sobre dicho software.

El software que la empresa posee hasta el momento se llama FIRESOFT, un software TPV desarrollado por la empresa FUTURA y que ofrece en varias versiones. Una versión ligera FIRESOFT OEM la cual únicamente contiene las opciones básicas orientadas a la pequeña empresa y a la gestión de ventas sin control de stock. La versión LITE añade la gestión de inventarios y la posibilidad de añadir "radiocomandas" (PDAs) además de otras funciones extra. Por último se encuentra la versión PRO. Esta versión es la más completa de todas e incluye gestión de proveedores, gestión multi-almacén, etc.

Entre las características principales de este software se encuentran:

- Separación de mesas
- Traspaso entre mesas
- Diferentes tarifas
- Información de mesas
- Diseño de tickets
- Informes de ventas
- Control de accesos
- Radiocomandas
- Gestión de inventarios
- Lector de códigos de barras
- Gestión de proveedores
- Gestión multialmacén
- Control de precios de coste

Figura 1: Interfaz de Firesoft

Sin embargo, no todo es positivo, ya que tiene ciertas cualidades indeseables, que son las que se pretenden mejorar.

Uno de los problemas más importantes es su falta de estabilidad. Mientras que con cargas bajas de trabajo el sistema suele funcionar bien, ante altas cargas de trabajo y el aumento de operaciones que se realizan de forma concurrente mediante las radiocomandas, las cuales aumentan considerablemente en ciertas fases de las jornadas, el sistema se vuelve inestable e inseguro, provocando caídas del sistema y desconexiones de los terminales portátiles que socaban el buen funcionamiento del sistema de trabajo; provocando servicios más lentos, pérdidas de información, y bloqueos de los pedidos que se encontraban en uso cuando se produjo la caída del sistema.

Otra de las características a mejorar es su conectividad. La capacidad para la conectividad de radiocomandas se limita únicamente a dos unidades, y lo deseable sería incrementar esa cifra hasta, como mínimo, seis unidades.

A su vez, se pretende mejorar el aspecto un tanto anticuado que muestra y simplificarla para hacerla más eficiente e intuitiva.

2. Estudio del arte

2.1.TPV

El TPV o POS (de las siglas en inglés Point of Sale) es la evolución del siglo XXI de la clásica caja registradora. En el mundo de la hostelería y venta al por menor de pequeños comercios los TPV son utilizados para registrar ventas, beneficios, pedidos, inventarios, historial de clientes, etc. En resumen, un TPV otorga control sobre las operaciones de negocio.

Un TPV básico consiste en una computadora, un cajón de dinero, una impresora de tiques, un monitor y dispositivos de lectura como lector de códigos de barras, teclados, etc. Los TPV registran las transacciones y permiten generar detallados informes, permitiendo tomar mejores decisiones comerciales. El TPV correcto permitirá mejorar la productividad y redundará en mejores ganancias.

2.1.1 Beneficios de usar un TPV

Un TPV puede revolucionar el funcionamiento de un comercio. Desde incrementar la productividad de los empleados hasta conocer si las inversiones obtienen beneficios.

2.1.1.1 Beneficios financieros

- Con el control y la monitorización sobre el inventario, un TPV puede ayudar a reducir drásticamente la pérdida del mismo, evitando que desparezca por robos, derroche o mal uso por parte de los empleados.
- Un TPV también asegura que cada producto es vendido por el precio correcto. Evitando que los empleados no introduzcan todo el dinero en la caja o que cobren de memoria, pudiendo equivocarse en la cantidad. Además, permite modificar los precios de los productos pulsando una tecla en el ordenador.
- Además, los detallados informes de ventas permiten modelar el funcionamiento de la empresa en función de los productos más vendidos, o que aportan mayor margen de beneficio, o promocionar los menos vendidos, etc. Es decir, permite tomar decisiones empresariales o comerciales en función del comportamiento del negocio a partir de los datos reales y en tiempo real que ofrece.

2.1.1.2 Información

- Información real, en tiempo real del estado del negocio. En cualquier momento del día se puede consultar información sobre las ventas hasta el momento, el dinero que hay en caja, beneficios hasta el momento, etc.
- Fácil gestión de inventario. Los informes de ventas permiten mantener el stock de producto correcto en cada momento, en función del estado del mismo y de las previsiones de ventas. Además, puede avisar cuando sea necesario realizar nuevos pedidos e incluso puede realizar el pedido automáticamente.
- Permite, también, construir una cartera de clientes recogiendo nombres, emails y direcciones. Esta información puede servir para realizar campañas de marketing, mejorar y focalizar los anuncios publicitarios en zonas geográficas, etc.

2.1.1.3 Mejoras productivas

- Mediante la automatización de actividades de aprovisionamiento de inventario así como la monitorización del mismo y el almacenamiento de la información de manera organizada que puede ser consultada en cualquier momento, un TPV puede reducir la necesidad de realizar tareas de papeleo repetitivas.
- Se incrementa la velocidad en las transacciones habituales. La realización de la factura del cliente mediante la lectura de los códigos de barras de los productos constituye una disminución del tiempo de servicio al cliente, lo que redunda en un mejor servicio. Así como, en el caso de la hostelería, el envío del pedido de los clientes desde el propio comedor a la cocina de forma telemática constituye también una disminución del tiempo de espera del cliente. La mejora productiva es sustancial.

2.1.2 Hostelería o venta al por menor.

El mercado de TPV está dividido en dos partes: venta al por menor (tiendas) y hostelería (restaurantes, cafeterías, hoteles, etc).

2.1.2.1 Venta al por menor.

De los dos grupos, el de venta al por menor tiene necesidades más simples en cuanto a TPV se refiere. Las transacciones se realizan de una vez, y es poco habitual que existan variaciones de los productos que venden.

Lo más habitual es que exista una matriz de productos en la que los productos se organizan en función de ciertas características (talla, modelo, color…). Por ejemplo, una matriz de productos permite tener un inventario y precio para un producto concreto, pero permite seleccionar la talla o el color en el momento de venta.
Otras necesidades básicas para estos comercios es la posibilidad de realizar ofertas (2x1, 3x2, 2ª unidad a menor precio, etc).

2.1.2.2 Hostelería.

Existe un tipo de TPV para cada tipo de establecimiento hostelero en función de las necesidades que precisa.
Un establecimiento que venda comida por peso necesitará un TPV con una interfaz para utilizar una báscula digital. Un restaurante se beneficiará de la reducción de los posibles errores producidos por notas manuscritas. Un restaurante de comida rápida reducirá el derroche de comida y mejorará la comunicación si tiene en la cocina una pantalla en que visualizar los pedidos de los clientes directamente desde el terminal de pedidos.
Además, en el caso de servicios restaurantes o establecimientos con servicios de terraza, deben tener la opción de tener varias cuentas abiertas al mismo tiempo, de modo que se pueda ir registrando el movimiento en cada mesa. Los hoteles tienen también requerimientos especiales, como la posibilidad de transferir cargos de servicios (restaurante, spa, etc) a la habitación del cliente.

En todas las configuraciones posibles de restaurante, un TPV amortiza su gasto con el registro del inventario y gastos generales. Si se usa y actualiza con regularidad, tras la

puesta en marcha con los primeros valores iniciales, el coste de mantenimiento se reduce drásticamente y siempre es beneficioso para cualquier negocio.

2.1.3. Hardware

2.1.3.1 Dispositivos de entrada

- Teclados y pantallas táctiles

Figura 2. Pantalla táctil típica de TPV

Figura 3. Teclado específico de TPV

Unas de las primeras decisiones que se deben tomar es si el TPV se necesita con pantalla táctil o teclado programable. Normalmente se utiliza pantalla táctil, sin embargo, los teclados programables son utilizados comúnmente en tiendas de comestibles, donde cada tecla se refiere a un producto.
Las pantallas táctiles son más intuitivas que los teclados y proporcionan una interfaz más flexible, así como una programación también más flexible.
Algunos sistemas usan teclados estándar, mientras que otros tienen teclados específicos, como el teclado plano de membrana de servicios de comida rápida, por ejemplo. En ocasiones los teclados de TPV incluyen un lector de banda magnética para gestionar el pago con tarjeta de crédito.
La elección de un sistema de entrada u otro, o ambos debe realizarse en función de las necesidades y el ambiente de trabajo al que se someterá el TPV.

- Lectores de códigos o escáneres

Figura 4. Lector laser de códigos de barras

Todos los escáneres funcionan básicamente de la misma manera. Leen el código de barras y envían el resultado al ordenador. Los lectores de código mejoran la velocidad y precisión en el proceso de facturación, evitando posibles errores humanos mediante la entrada de los códigos manualmente y agilizando el proceso al ser necesario, únicamente, que el empleado dirija el lector hacia la etiqueta del producto. Para seleccionar uno en concreto ha de tenerse en cuenta el ámbito de trabajo.
Los escáneres más básicos son baratos, pero tienen un rango muy corto, lo que requiere que los códigos a escanear se coloquen a un máximo de 3 pulgadas.

A continuación se encuentran los escáneres laser, que usan un rayo de luz para leer los códigos, que pueden escanear a mayor distancia. Este tipo de escáner es el más utilizado en la venta al por menor por ser muy flexible, fácil de usar y preciso.
Los escáneres omnidireccionales permiten escanear el objeto desde cualquier ángulo. Son los comúnmente usados en supermercados.

- Terminales de mano

Figura 5. PDA clásico de TPV

El dispositivo de entrada más moderno es el terminal de mano con conectividad wireless. Esencialmente una PDA. Cada terminal transmite pedidos hacia la estación base. Supone una ventaja importante al permitir incrementar el tiempo que el empleado se encuentra en la terraza interactuando con los clientes, al no tener que volver al terminal fijo a introducir los pedidos.
Los terminales de mano son más caros que los terminales tradicionales con pantalla táctil. Sin embargo, pueden amortizar el coste reduciendo el tiempo de espera de los clientes y, por tanto, mejorando la productividad y la calidad del servicio.
Los nuevos TPV incluirán aplicaciones para tablets y smartphones.

2.1.3.2 Hardware adicional

- Impresoras

Figura 6. Impresora térmica

Todo TPV necesita al menos una impresora para generar facturas y recibos para los clientes. Además, muchos restaurantes utilizan también impresoras para visualizar los pedidos en la cocina.
Se suelen emplear impresoras térmicas que utilizan calor y papel térmico especial para generar recibos. Son más caras pero mucho más rápidas, silenciosas y fiables.

- Cajón de dinero

Figura 7. Cajón de dinero TPV

Los cajones de dinero son esenciales para guardar de forma segura el dinero, recibos de tarjetas de crédito, vales de regalo y cualquier otro tipo de papeleo importante. Lo más importante a la hora de elegir el cajón de dinero es la robustez.
En la mayoría de los cajones de dinero, la señal de apertura del cajón proviene de la impresora de tiques. Comprar el sistema completo al mismo fabricante asegura compatibilidad y facilidad para encontrar repuestos.

- Pantallas de cliente.

Figura 8. Display cliente TPV

Este periférico muestra al cliente información sobre el producto y el precio del mismo. Algunos pueden incluso mostrar anuncios publicitarios.

- Escáner de huellas

Figura 9. Lector de huella dactilar

Limitar el acceso de los empleados al terminal TPV es algo crítico. Los métodos más comunes son simples contraseñas o tarjetas magnéticas, pero dichos métodos están sujetos a fallos de seguridad: las contraseñas pueden ser compartidas o espiadas por parte de otros usuarios, y los empleados pueden perder o robar tarjetas a otros empleados. El nuevo sistema de seguridad que ha surgido para suplir esas carencias es el uso de pequeños lectores de huellas dactilares, este lector asegura que el empleado accederá al sistema con los correctos privilegios y nadie podrá suplantarlo.

- Otros dispositivos

Existen otros muchos diversos dispositivos que pueden añadirse o estar integrados dentro de un TPV tales como: básculas, teclados de seguridad (introducir credenciales), etc…
Cualquier dispositivo que pudiese ser utilizado para gestionar los procesos de gestión de ventas podrían ser integrados, por lo tanto, la lista de periféricos existente es amplia y abierta, es decir, pueden incorporarse periféricos específicos en función de las necesidades del negocio.

2.1.4 Software

La funcionalidad básica de un TPV no varía mucho de uno a otro. Sin embargo, características especiales incrementan el coste. Por lo tanto, se hace esencial conocer y comprender las necesidades del negocio y que características debe tener el TPV para incrementar la eficiencia.

Conocer el negocio, los factores que lo hacen único, si existen métodos de venta poco usuales, si las ventas son muy detalladas, etc. Son cosas que un programador debe conocer para realizar un buen software.

2.1.4.1 Características básicas

La mayoría de los TPV cumplen un largo número de funciones comunes: mostrar productos y precios de una venta, manejo de tasas, devoluciones, diferentes opciones de pago, descuentos, informes de cuentas, registro de inventario, etc.

El software de los TPVs de hostelería permiten crear cuentas por pedido o mesa, pedidos especiales, registrar pedidos por camarero, mover pedidos de barra a mesa, lista de espera, etc.

2.1.4.2 Acceso remoto

Algunos TPVs ofrecen la posibilidad de acceder a los datos del día a través de una conexión a internet desde cualquier lugar, a través de la web. Las aplicaciones para Smartphone y tablets se irán extendiéndo y permitirán realizar este seguimiento desde cualquier lugar.

2.1.4.3 Programas de puntos

Los TPV pueden ayudar a gestionar un programa de puntos a los pequeños restaurantes, cosa que hasta ahora solo se daba en las grandes cadenas de restaurantes. Este tipo de programas de fidelidad de clientes, que premia a los clientes con incentivos o descuentos, están creciendo en popularidad cada vez más.

2.1.4.4 Otras características

Visores que muestren imágenes actuales del producto que los clientes compran. Esto ayuda a los clientes a reconocer el producto que quieren.
Verificación de edad. Automáticamente se pregunta a los clientes su edad para compras como alcohol o tabaco.

Registrar información de los clientes como número de teléfono, e-mail o dirección, un histórico de compras, etc. Esta información permite dar un servicio más personalizado al cliente.

También pueden implementar un servicio de "e-comercio", permitiendo realizar pedidos o compras online.

2.1.5 Soporte

Un colapso del TPV, aunque poco común (los buenos TPV ofrecen una muy buena estabilidad), puede provocar clientes insatisfechos, pérdida de beneficios y considerables dolores de cabeza al gerente del negocio. Por lo tanto, un buen soporte se convierte en un componente indispensable dentro del TPV.

Las características del soporte que serán necesarias para cada TPV varían en función del ámbito de aplicación de dicho TPV. En algunos ámbitos puede ser crítico el tener el TPV caído, mientras que en otros tal vez exista cierta tolerancia hacia dicha eventualidad. A más completo soporte, mayor coste.

Algunas características a tener en cuenta son:

2.1.5.1 Soporte telefónico 24x7

Una de las primeras vías a agotar cuando surge algún problema es la resolución telefónica. Es posible, en ocasiones, que el fallo o error sea común y fácilmente subsanable mediante algún parámetro de configuración. En estos casos, el realizar el diagnóstico de forma telefónica permite ahorrar tiempo y molestias.

Por ello, es frecuente contar con un servicio de soporte activo 24 horas 7 días a la semana. Ya que el horario de mayor actividad de algunos negocios está fuera del horario normal de oficina.

Sin embargo, si el negocio propietario del TPV tiene un horario normal de oficina (no se utiliza de madrugada, ni la mayoría de festivos), con un servicio normal de horario de oficina bastaría y se conseguiría reducir el coste de mantenimiento.

2.1.5.2 Servicio técnico local

Para los problemas que no pueden ser resueltos por vía telefónica se ha de contar con un servicio técnico que pudiera acudir a solucionar el problema. Un servicio técnico local asegura un tiempo de respuesta mucho menor.

Normalmente, si no se puede solucionar el problema 'in-situ' cuentan con un servicio de préstamo de equipo para poder mantener la actividad empresarial mientras el equipo averiado es reparado.

La mayor disponibilidad del servicio técnico incrementa el coste de mantenimiento, por lo tanto, sería conveniente ajustar lo máximo posible la disponibilidad del servicio técnico en función de las necesidades del negocio.

2.1.5.3 Asistencia remota

En caso de no contar con un servicio técnico local, es recurrente también, el uso de herramientas telemáticas para tratar de solucionar los problemas. Lo más habitual es establecer un túnel de acceso para acceder y controlar el equipo afectado de forma remota.

Otra solución al problema de no contar con un servicio técnico local, a la que también se suele recurrir, es el envío de un repuesto inmediatamente después del reporte de la avería. En el momento en que llega el repuesto, el equipo averiado se envía para su reparación.

2.1.5.4 Aprendizaje

Para sacarle todo el partido posible al TPV es necesario que tanto los administradores, como todos los empleados conozcan a fondo su funcionamiento. Conocer en profundidad como utilizar el TPV permite desarrollar mayores velocidades de operación, resolver pequeños desajustes de configuración, y obtener datos más acordes a la realidad.
Si por el contrario, los usuarios no tienen una correcta formación sobre el funcionamiento del PTV pueden ocasionar problemas de configuración, introducir datos en la base de datos de forma incorrecta o incompleta, producir pérdidas de información válida, etc.
Poner a disposición de los usuarios herramientas de ayuda como manuales, FAQs, tutoriales o guías, así como ofrecer clases o cursos a los trabajadores de las empresas, o al menos al administrador, se hacen indispensables para aprovechar cuanto antes el máximo potencial que la herramienta pueda ofrecer.

2.2.Desarrollo software para TPV

Debido al tipo de aplicaciones que conforma el software TPV, cuyos requisitos o funciones mínimas básicas son bien conocidas y generales, así como el ámbito general de utilización de las mismas, es usual, por parte de los programadores, recurrir a herramientas, lenguajes, o formas de programar (orientado a objetos, eventos, etc…), que se ajustan y facilitan la programación de dichas aplicaciones.

Una de las principales herramientas empleadas por los programadores para implementar TPVs es Visual Fox Pro.
Visual Fox Pro es un lenguaje de programación basado en procedimientos y orientado a objetos. Pertenece a la familia xbase, por lo que su programación es sencilla y estructurada, y, por lo tanto, fácil de entender y aprender tanto para programadores principiantes como expertos. Además, integra un sistema gestor de bases de datos, lo que lo convierte en una importante herramienta para la programación de aplicaciones que requieran de utilizar una base de datos.
Es un lenguaje con un rendimiento en términos de velocidad muy altos. Sin embargo, esta velocidad se consigue mediante el uso de la memoria, lo que se puede convertir en una desventaja, pues requerirá mucha memoria RAM.

Se ha decidido no realizarlo utilizando esta herramienta debido a que Microsoft anunció que Visual Fox Pro no obtendría nuevas versiones a partir de 2007, y que el soporte para esta herramienta también desaparecerá a partir de 2015, por la dificultad que supone adaptarla a sistemas de 64 bits. Por esto, a pesar de tener una amplia comunidad que continua desarrollando proyectos utilizando esta herramienta, se considera una herramienta en plena obsolescencia, y no resulta una herramienta idónea para desarrollar aplicaciones con vistas a futuro.

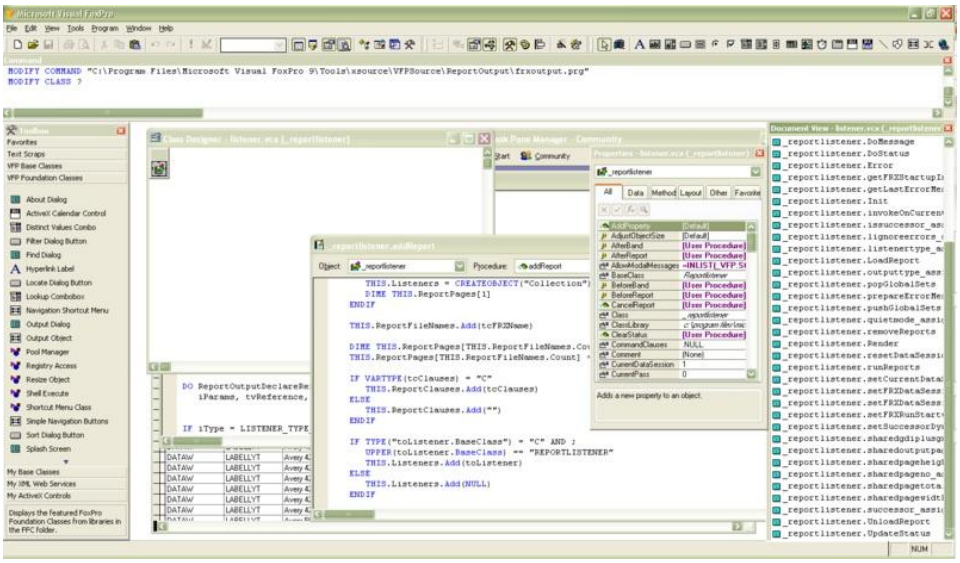


Figura 10: Visual Fox Pro

2.3. Opciones de software que ofrece el mercado.

El mercado de software destinado a realizar funciones de TPV es muy amplio y variado. Existen muchas opciones disponibles en función del negocio y sus necesidades. Algunas empresas ofrecen paquetes de software multisectorial que mediante la correcta configuración logra un grado de adaptación aceptable. Otros sin embargo, ofrecen una base software sobre la que asentar los diferentes módulos funcionales en función de las necesidades del cliente. Existen también los que únicamente están destinados para un tipo de negocio concreto y ofrecen pobres opciones de configuración.
A continuación se presenta una pequeña selección de distintos sistemas software tanto de pago como gratuitos, y para diferentes plataformas:

- **InterTPV**

Sistema TPV de pago, para plataformas Windows. Es una aplicación diseñada para trabajar en red y mediante código de barras. Presenta, además de las características básicas de TPV, la posibilidad de elaboración de presupuestos y facturas, y está preparado para modo de pantalla normal o pantalla táctil.
Se destacan las siguientes funciones de la aplicación: control de acceso y dependientes, impresión de etiquetas con código de barras simple y doble, completo cierre de caja desglosado en efectivo y tarjeta, impuestos diferenciados en ventas y compras, y numerosos informes y estadísticas.

- **123 TPV Net**

Es un sistema gratuito válido para la plataforma Windows. Permite tener control de ventas, stock, empleados, cierres diarios, etc.
Las principales características son: gestión de productos, proveedores, clientes, formas de pago, secciones, ventas (código de barras o táctil), almacenes (stock y pedidos a proveedores), configuración, tarifas, promociones, estadísticas, envíos de sms a clientes, deudas de clientes y albaranes.

- **Gestión TPV**

Se trata de un sistema gratuito y online. Al ser online, únicamente con conexión a internet y un navegador se puede acceder, lo que lo hace independiente de la plataforma y accesible las 24 horas del día desde cualquier lugar. No requiere ningún tipo de instalación.
Sus características principales son: gestión de compras, ventas, devoluciones, artículos, clientes y proveedores; emisión, impresión, y envío por e-mail de facturas, albaranes, pedidos, presupuestos y tickets; histórico de operaciones de compra y venta; ilimitados almacenes y sucursales; multipuesto; clasificación de productos en familias y subfamilias; diferentes impuestos; múltiples formas de pago; control de stock; soporte para pantalla táctil y lector de código de barras; gestión de privilegios por usuarios o grupos; copias de seguridad automáticas por día; y ayuda en línea.

- **Mi tienda**

Mi tienda es un sistema de pago, para plataformas Windows que ofrece un sistema de cómputo para administrar y controlar las cajas de manera sencilla.
Características principales: registro de la venta diaria, manejo de diferentes divisas, aceptación de pagos sin cerrar las cuentas, registro inmediato de nuevos productos, entradas y salidas de productos sin salir del sistema, manejo de ofertas, diferentes listas de precios, diferentes formas de pago, manejo de código de barras y control de ventas por vendedor y por cliente.

- **SimpleTPV**

Es un sistema TPV para plataformas Windows, de pago. Su filosofía se basa en la simplicidad para no sobrecargar la aplicación con funciones complejas, costosas y caras. Por ello, es una solución algo más barata, al limitar su uso a: controlar un almacén, vender y cobrar.
Sus características principales son: creación y gestión de clientes, artículos, formas de pago y dependientes; entrada de artículos y venta de los mismos; cierre de caja con arqueo ciego o no; listados de clientes, artículos e historial de ventas; y creación de códigos de barras propios.

- **POSper**

Es un sistema diseñado para PyMES, para plataformas UNIX. Ofrece soporte para una amplia variedad de hardware y bases de datos. Cuenta con soporte para touch screen, diseño orientado para negocios al por menor y restaurantes, opciones de tarjetas de cliente, un sistema de descuentos con listas de precios para clientes, modificadores para despuestos, impuestos sobre el cambio y procesamiento de órdenes con notas de entrega, entre otras características.

- **Lemon POS**

Es un software de punto de venta de código abierto. Emplea MySQL como motor de base de datos y puede ser utilizado tipo cliente/servidor con una base de datos central y varios terminales TPV en la red local.
Posee una interfaz moderna y personalizable, un panel de búsqueda, herramienta de verificación de precios, herramientas administrativas, impresión de reportes, etc.

- **FloreantPOS**

Es una aplicación TPV independiente de la plataforma, construido en JAVA, con soporte touch screen diseñado para todo tipo y tamaño de restaurantes. Ofrece manejo de entregas, impuestos, descuentos, agrupamiento de alimentos, ticket de cocina, sistema de pago combinado y reporte de ventas. Está liberado bajo licencia MRPL.

- **LibrePOS**

Se trata de un sistema disponible para las plataformas Windows, Mac, Solaris y Linux. El sistema está internacionalizado y soporta diferentes idiomas: español, inglés, alemán, portugués, italiano, etc.
El sistema está diseñado para poder interactuar mediante pantalla táctil, sin necesidad de ratón o teclado. Para los restaurantes incluye características muy interesantes como la visualización de las mesas en el bar y su información.
La características principales son: gestión de los cierres de caja, los productos, las categorías de los productos; soporta impresoras ESC/POS, visores de cliente y lector de código de barras; gestión de permisos de usuario; edición de productos; e informes de ventas.

2.4.Bases de datos

Las bases de datos conforman la principal herramienta de almacenamiento y gestión de información en el mundo de la informática y el software.
Ofrecen una herramienta muy potente para almacenar información de forma estructurada y organizada, y, a la vez, permite realizar búsquedas de información de una manera muy eficiente y rápida.

A continuación, se detallan los diferentes tipos de bases de datos que podemos diferenciar en función de diferentes formas de clasificarlas:

2.4.1.En función de la variabilidad de los datos almacenados

Esta clasificación diferencia a las de datos en función de si ser permite o no la variación de los datos una vez introducidos.
Se distinguen:

2.4.1.1.Bases de datos estáticas

Las bases de datos estáticas son bases de datos de sólo lectura, es decir, no permiten realizar modificaciones de los datos una vez introducidos. Únicamente permite realizar operaciones de consulta de datos.
Su utilización fundamental consiste en almacenar datos históricos.

2.4.1.2.Bases de datos dinámicas

Estas bases de datos en las que la información almacenada puede ser modificada, actualizada o borrada.
Permite realizar operaciones de escritura, actualización y borrado, además de las operaciones de consulta.

Este es el tipo de base de datos que se puede encontrar en cualquier supermercado, videoclub, etc…

2.4.2.En función del modelo de administración de datos

2.4.2.1.Bases de datos jerárquicas

En este modelo de administración se utilizan árboles para la representación lógica de los datos. El nodo que no tiene padres se denomina raíz, y el los nodos que no tienen hijos se denominan hojas.
Estas bases de datos se emplean, sobre todo, para aplicaciones que manejan grandes cantidades de datos muy compartidos.

2.4.2.2.Bases de datos de red

Este modelo es muy similar al de las bases de datos jerárquicas. Su diferencia radica en el hecho de que un nodo pueda tener varios padres, cosa que no ocurría en el modelo jerárquico. Y a su vez, pueden existir relaciones entre nodos hermanos, otra característica no contemplada en el modelo jerárquico.

2.4.2.3.Bases de datos relacionales

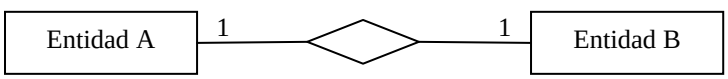
Las bases de datos relacionales son las más utilizadas en la actualidad para la gestión de bases de datos. Se fundamentan en el modelo entidad-relación.

El concepto entidad, empleado por este modelo, hace referencia a objetos que tienen un significado real. Estas entidades constituyen tablas de la base de datos, cuyos elementos serían las tuplas o filas, y cuyos atributos serían las distintas columnas de la tabla.
Por ejemplo, una entidad, dentro del entorno que ocupa a este proyecto, podría ser la entidad producto. Y los atributos de esta entidad serían el nombre, el precio, la familia, etc. Un elemento de la tabla productos podría ser una botella de agua, y sus atributos el propio nombre, su precio, la familia de productos a la que pertenece, etc.

Las entidades no están aisladas, sino que se encuentran relacionadas entre sí. Con esto se consigue evitar la existencia de redundancia en la base de datos y por lo tanto la ineficiencia en cuestiones de espacio que ocupan los datos. Estas relaciones pueden ser de tres tipos:

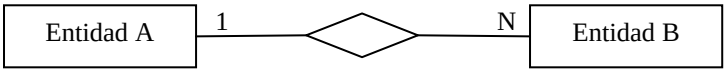
- Relación 1 a 1.

Esto implica que un elemento de una entidad se relaciona únicamente con un único elemento de otra entidad.



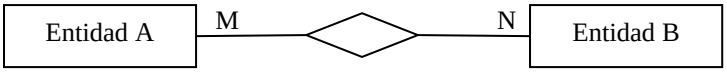
- Relación 1 a varios.

En este tipo de relaciones, un elemento de una entidad se puede relacionar con varios de elementos de otra entidad.



- Relación varios a varios.

En este caso, varios de elementos de una entidad se pueden relacionar con varios de otra entidad.



3. Caso de estudio

Como se ha comentado anteriormente, este proyecto consiste en el desarrollo de la primera parte de un proyecto mayor. Este proyecto mayor tiene la estructura como se ve a continuación.

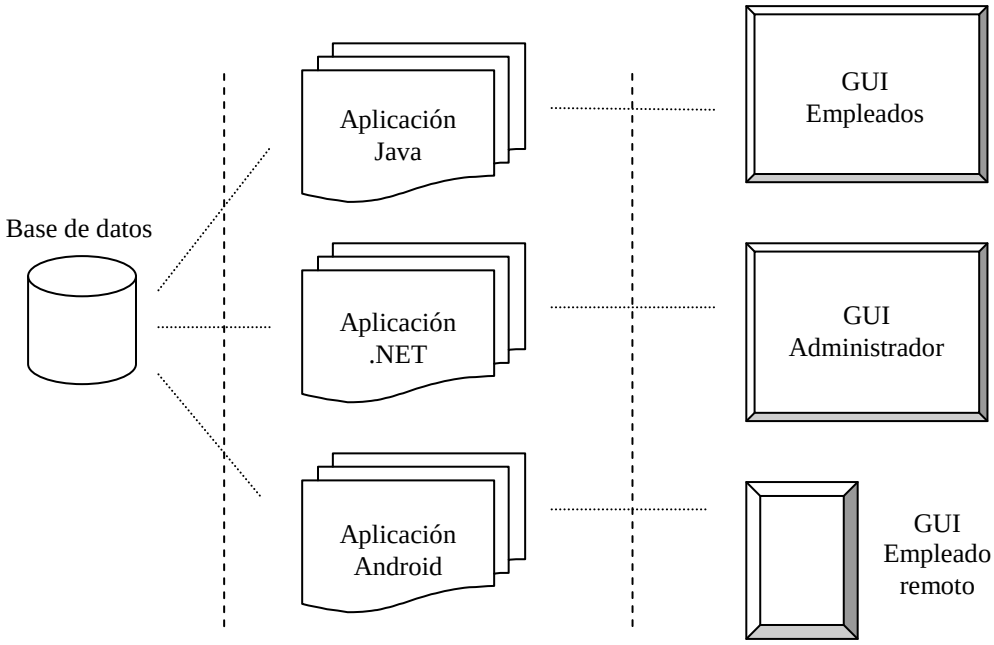


Figura 11. Arquitectura 1

Se desarrollará la primera parte, consistente en el diseño e implementación de la base de datos y la aplicación Java con la interfaz de empleados.

3.1. Análisis

3.1.1 Identificación de actores.

Se distinguen dos tipos de actores:

- Empleados
- Administradores

3.1.1.1 Empleados.

Se considerará empleado a cualquier usuario de la aplicación que no conozca la contraseña de acceso a las zonas restringidas.

3.1.1.2 Administradores.

Se considerará administrador a cualquier usuario de la aplicación que conozca y pueda utilizar una contraseña de acceso a las zonas restringidas en caso de ser necesario.

Administrador		1. Configuración de la aplicación
		2. Movimientos de caja
		3. Cerrar aplicación
	Empleado	4. Abrir editar o generar pedido actual
		5. Modificar productos de pedido actual
		6. Modificar empleados de pedido
		7. Imprimir ticket
		8. Invitar
		9. Juntar pedido
		10. Cobrar
		11. Cancelar cobro
		12. Abrir jornada de empleado
		13. Cerrar jornada de empleado

3.1.2 Identificación de los casos de uso.

El punto inicial de todos los casos de uso, salvo que se exprese específicamente será desde el panel principal de la aplicación, que muestra el grid de mesas.

3.1.2.1 Casos de uso disponibles para administrador

1. **Configuración de la aplicación**
   Objetivo: Acceder al menú de configuración del terminal.
   Actores: Administrador.
   Pasos:
   Administrador: Pulsa botón de opciones.
   Terminal: Muestra el menú de opciones.
   Administrador: Pulsa el botón de configuración.
   Terminal: Muestra el panel de introducción de contraseña.
   Administrador: Introduce contraseña y pulsa "Ok".
   Terminal: Valida la contraseña.
   Terminal: Muestra el menú de configuración.
   Administrador: Establece los parámetros de configuración y aplica cambios.
   Terminal: Guarda los cambios y vuelve al menú de opciones.
   Extensiones:
   Terminal: En caso de contraseña incorrecta muestra un panel de error y vuelve a mostrar el menú de opciones.

   Diagrama de actividad:

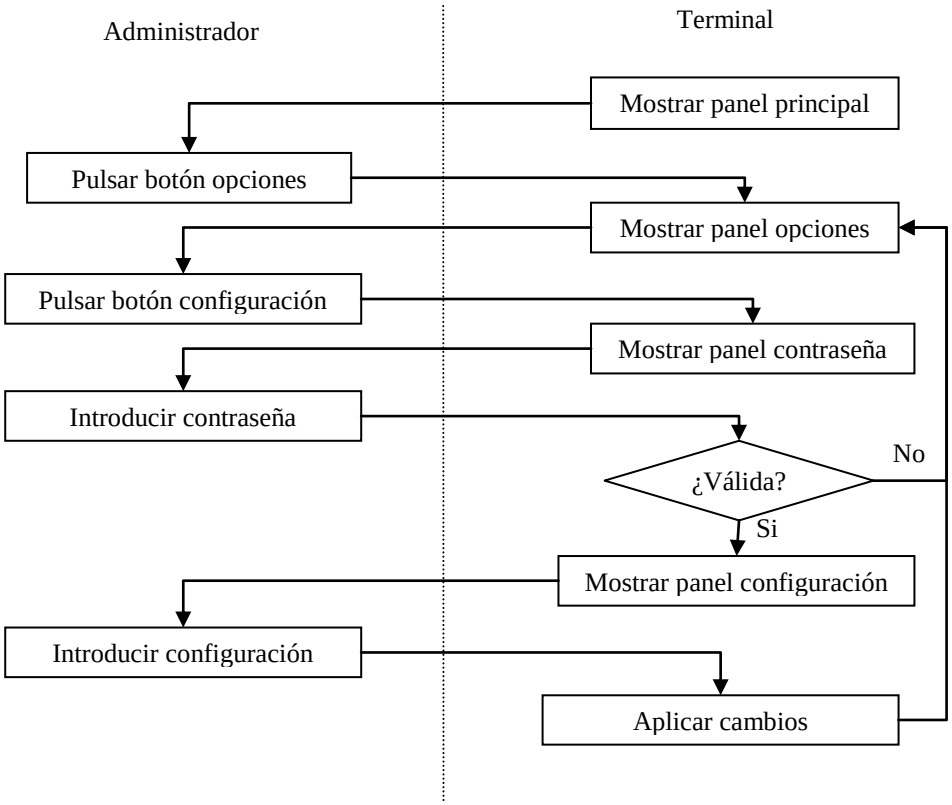


Diagrama 1: Configuración

2. **Movimientos de caja**

   Objetivo: Registrar los ingresos o extracciones de efectivo en caja por motivos administrativos.

   Actores: Administrador.

   Pasos:

   Administrador: Pulsa botón de opciones.
   Terminal: Muestra el menú de opciones.
   Administrador: Pulsa el botón "El jefe".
   Terminal: Muestra el panel de introducción de contraseña.
   Administrador: Introduce contraseña y pulsa "Ok".
   Terminal: Valida la contraseña.
   Terminal: Muestra el menú de movimientos de caja.
   Administrador: Introduce la cantidad y pulsa ingresar o extraer.
   Terminal: Procesa la operación y vuelve a mostrar el menú de opciones.

   Extensiones:

   Terminal: En caso de contraseña incorrecta muestra un panel de error y vuelve a mostrar el menú de opciones.

   Diagrama de actividad:

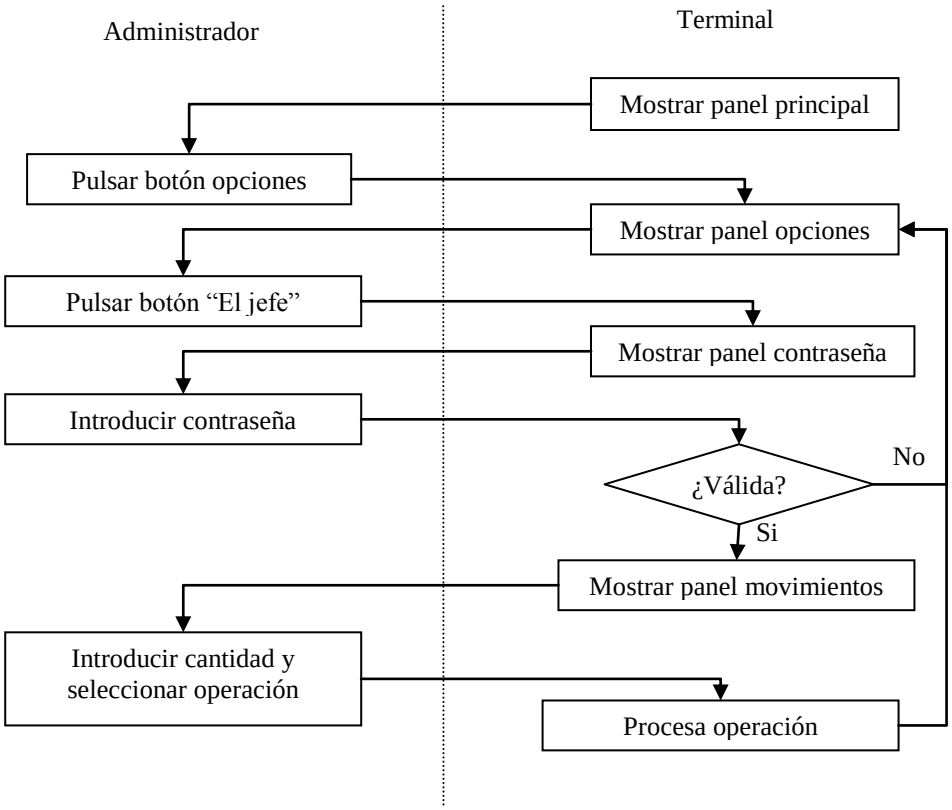


Diagrama 2: Movimientos de caja

3. Cerrar aplicación

Objetivo: Cerrar la aplicación.
Actores: Administrador.
Pasos:

Administrador: Pulsa botón de opciones.
Terminal: Muestra el menú de opciones.
Administrador: Pulsa el botón de salir.
Terminal: Muestra el panel de introducción de contraseña.
Administrador: Introduce contraseña y pulsa “Ok”.
Terminal: Valida la contraseña.
Terminal: Cierra la aplicación.

Extensiones:

Terminal: En caso de contraseña incorrecta vuelve a mostrar el menú de opciones.

Diagrama de actividad:

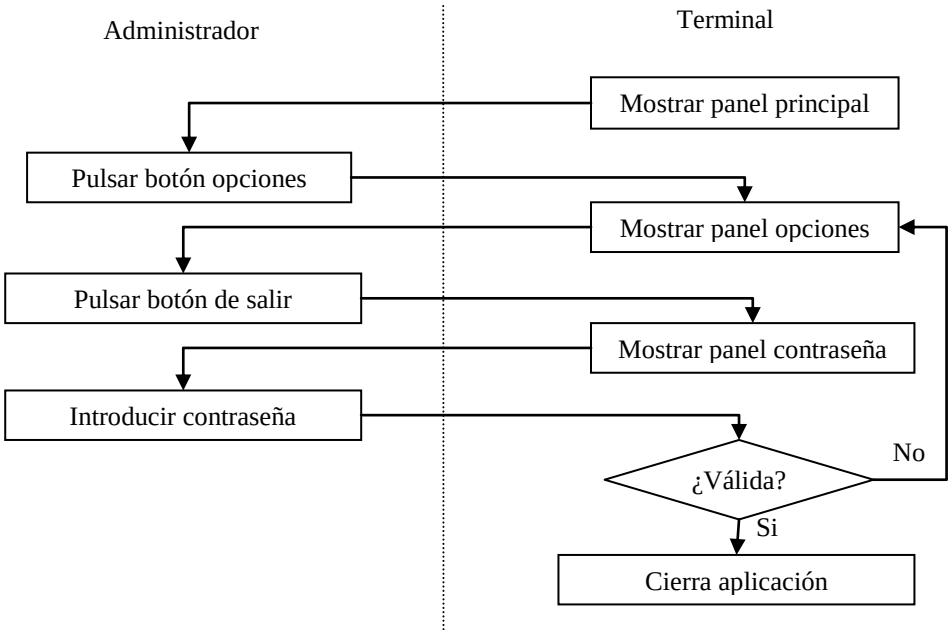


Diagrama 3: Salir

3.1.2.2 Casos de uso disponibles para empleados.

1. **Abrir, editar o generar pedido actual.**

   Objetivo: Mostrar el panel de visualización de mesas con los datos del pedido actual de la mesa seleccionada cargados. En caso de no haber pedido actual en la mesa seleccionada se generaría un nuevo pedido.
   Datos y contenido del pedido podrán ser editados.
   Actores: Empleados y administradores
   Pasos:

   Actor: Pulsa botón de una mesa o barra situado en el grid de mesas o barras, respectivamente.
   Terminal: Muestra el panel de contenidos de mesa y carga la información del pedido actual en el mismo, en caso de no existir pedido actual genera un nuevo pedido vacío.
   Actor: Edita datos del pedido, añadiendo o borrando artículos o modificando empleados responsables.
   Terminal: Muestra los cambios producidos por el actor.
   Actor: Pulsa el botón volver.
   Terminal: Muestra el panel principal.

   Extensiones:

   I.

   Terminal: En caso de que el pedido a actual a abrir no tenga empleados asignados, muestra un menú de selección de empleados.
   Actor: Pulsando en cualquiera de los empleados del menú de selección de empleados se asignarán al pedido.

   II.

   Actor: Introduce una cifra en el panel principal y pulsa el botón correspondiente al número.
   Terminal: se abrirá la mesa indicada por el número correspondiente a la cifra introducida.
   Terminal: en caso de no existir un botón de mesa con el número introducido se generará uno nuevo y se añadirá al panel inferior.

III.

Situación inicial: Panel de visualización de mesa.
Actor: Introduce un número entero y pulsa botón de barra o terraza.
Terminal: Carga en el panel los datos correspondientes al pedido actual de la mesa con el número introducido. En caso de no existir, genera uno vacío.

Diagrama de actividad:

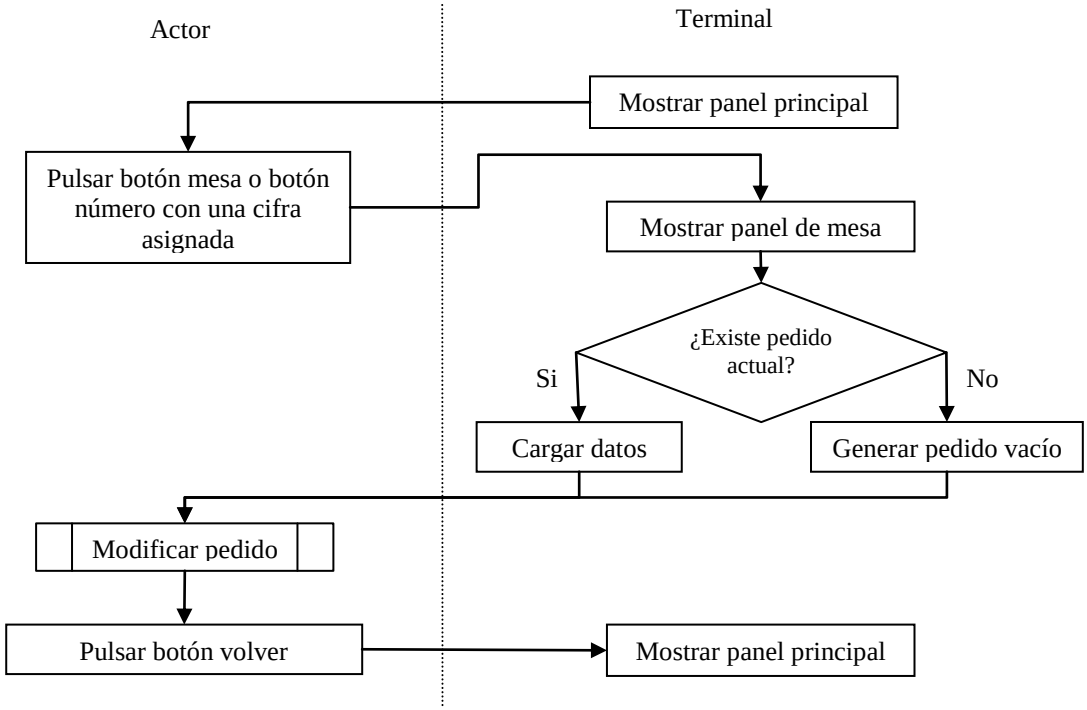


Diagrama 4: Abrir, editar o generar pedido

2. Modificar productos de pedido actual.

Objetivo: Modificar datos de los productos asignados al pedido.
Actores: Empleados y administradores.
Situación inicial: Para poder modificar un pedido se debe encontrar en el panel de visualización de mesas.
Pasos:

Actor: Pulsa botón de familias de productos para modificar la familia de productos disponibles para incluir.
Terminal: Muestra el panel de productos correspondientes a la familia seleccionada.
Actor: Pulsa en el producto que se quiera añadir al pedido.
Terminal: Muestra los productos introducidos.
Se repite el proceso tantas veces como sea necesario.

Extensiones:

I.

Actor: Introduce una cifra entera y posteriormente pulsa en un producto.

Terminal: Añadirán la cantidad producto indicado por la cifra introducida.

II.

Actor: Selecciona un producto de la lista, introduce una cifra entera y pulsa el botón unidades.

Terminal: Establece la cantidad indicada al producto seleccionado.

III.

Actor: Selecciona un producto de la lista, introduce una cifra y pulsa el botón precio.

Terminal: Establece la cifra introducida como precio unitario del producto seleccionado.

IV.

Actor: Selecciona un producto y pulsa el botón borrar.

Terminal: Decrementa la cantidad de producto seleccionado en una unidad. Si no se selecciona previamente el producto, se considera seleccionado el último producto de la lista.

Diagrama de actividad:

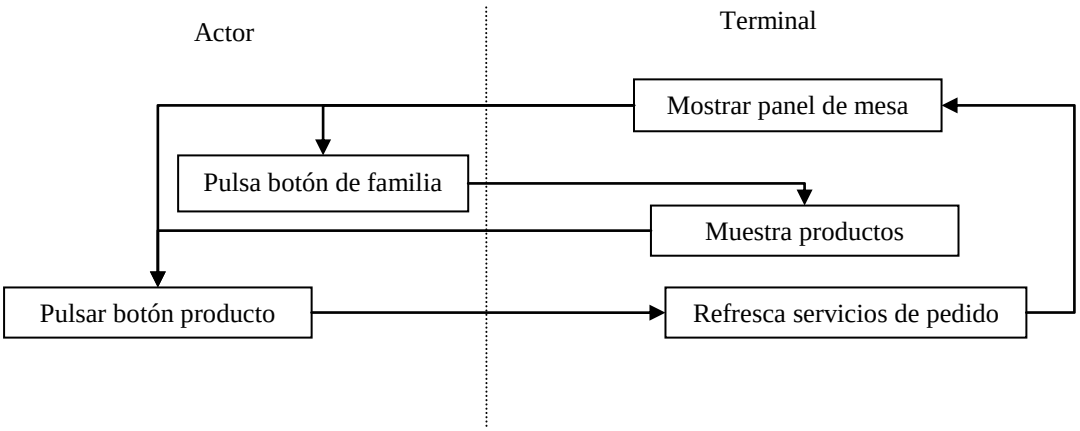


Diagrama 5: Modificar pedido actual

3. Modificar empleados de pedido.

Objetivo: Modificar los camareros asignados responsables de un pedido.

Actores: Empleados y administradores.

Situación inicial: Panel de visualización de mesas con el pedido que se quiera modificar cargado.

Pasos:

Actor: Pulsa botón del empleado a modificar.

Terminal: Muestra el panel de selección de camareros o dependientas (en función del botón pulsado).

Actor: Pulsa en el empleado que se quiera asignar al pedido.

Terminal: Asigna el empleado al pedido y refresca la visualización de los datos.

Diagrama de actividad:

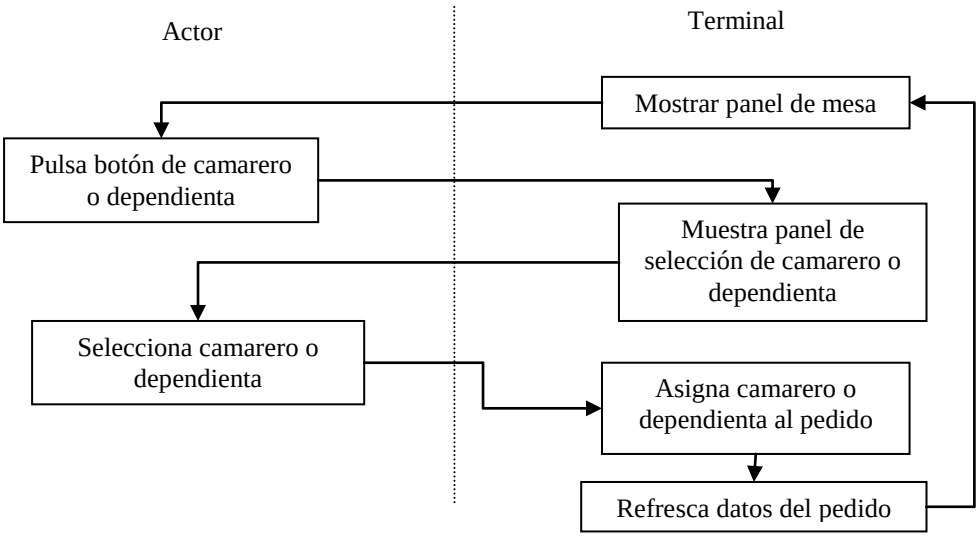


Diagrama 6: Modificar empleados de pedido

4. Imprimir ticket.

Objetivo: Imprimir los tickets que sirven como factura de los clientes con la información del pedido correspondiente.

Actores: Empleados y administradores.

Situación inicial: Panel de visualización de mesas con el pedido que se quiera modificar cargado.

Pasos:

Actor: Pulsa botón de imprimir.

Terminal: Imprime el ticket.

Extensiones:

I.

Actor: Pulsa el botón cobrar.

Terminal: Cobra la mesa e imprime un ticket donde se refleja la cantidad entregada por el cliente y el cambio.

II.

Actor: Pulsa tecla ticket.

Terminal: Si está en estado "Si" pasará al estado "No" y viceversa. En el estado "No" del ticket no se imprimirá ticket cuando se realice un cobro (usado para ahorrar papel cuando se entrega el dinero justo).

Diagrama de actividad:

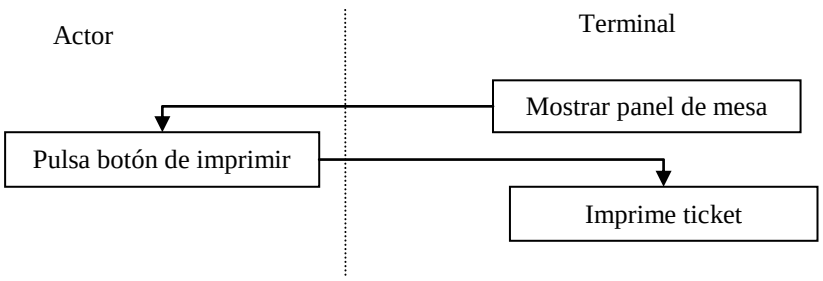


Diagrama 7: Imprimir

5. Invitar.

Objetivo: Añadir los pedidos de invitación a la tabla de invitaciones.
Actores: Empleados y administradores.
Situación inicial: Panel de visualización de mesas con los datos referentes al pedido actual cargados.
Pasos:

Actor: Pulsa el botón de invitar.
Terminal: Establece el pedido como cobrado y añade el pedido a la lista de invitaciones.
Terminal: Se genera un nuevo pedido actual vacío.

Diagrama de actividad:

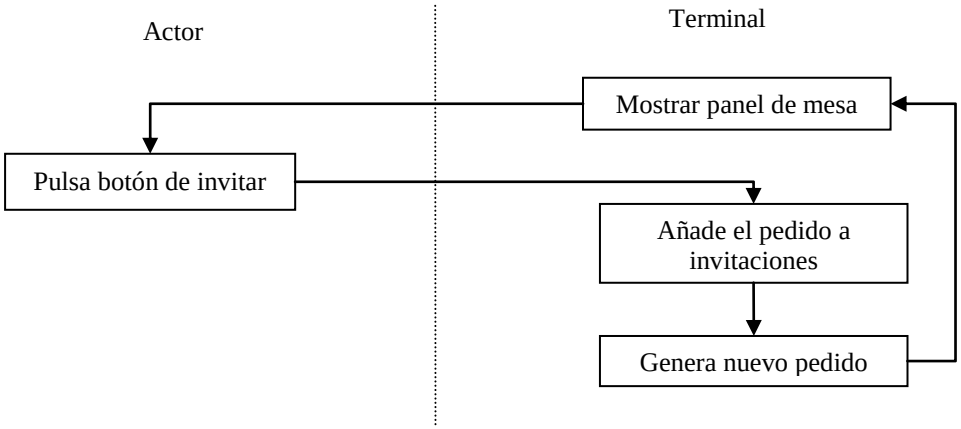


Diagrama 8: Invitar

6. Juntar pedidos.

Objetivo: Juntar pedidos que se encuentran distribuidos en varias mesas. También sirve para mover un pedido de una mesa a otra que se encuentre vacía.
Actores: Empleados y administradores.
Situación inicial: Panel de visualización de mesas con los datos referentes al pedido actual cargados.
Pasos:

Actor: Pulsa el botón de juntar.
Terminal: Muestra el grid de mesas.
Actor: Pulsa sobre una de las mesas.
Terminal: Muestra el panel de visualización de mesas con los datos de los pedidos anterior y actual en la mesa actual.

Diagrama de actividad:

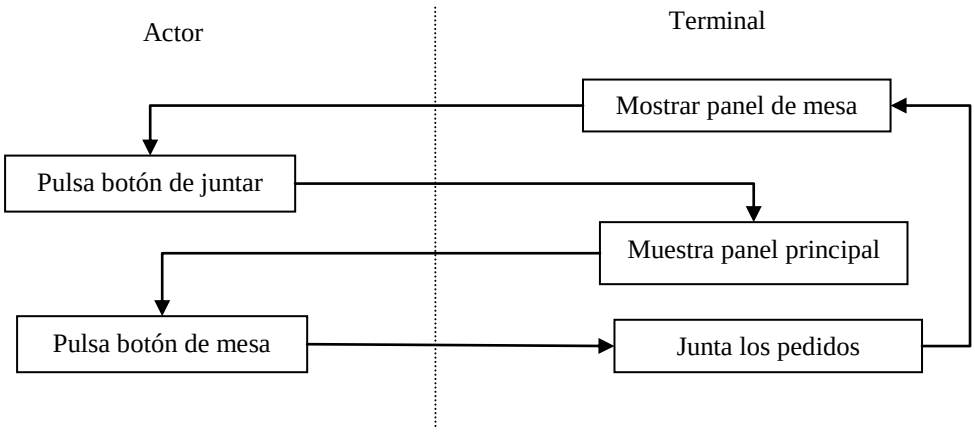


Diagrama 9: Juntar

7. **Separar cuenta.**

Objetivo: Separar las cuentas por productos, permitiendo con ello pagar por separado a los clientes.

Actores: Empleados y administradores.

Situación inicial: Panel de visualización de mesas con los datos referentes al pedido actual cargados.

Pasos:

Actor: Pulsa el botón de separar.

Terminal: Muestra el panel de separar cuenta y genera un nuevo pedido auxiliar.

Actor: Selecciona el producto a mover y pulsa sobre las flechas para mover los productos de una lista a otra.

Terminal: Elimina de la lista de productos del pedido actual el servicio indicado y lo añade al pedido auxiliar. Actualiza los datos de las listas.

El actor puede realizar esta operación tantas veces como sea necesaria para configurar el pedido auxiliar.

Actor: Introduce la cantidad entregada por el cliente y pulsa cobrar.

Terminal: Cobra el pedido que se haya seleccionado.

Actor: Cierra el panel de separar cuenta.

Terminal: En caso de que quede algún servicio sin cobrar recompone el pedido y la lista de servicios adecuadamente.

Extensiones:

I.

Terminal: En caso de que el importe total sea mayor que la cantidad entregada muestra un mensaje de error y vuelve al panel de separar cuenta.

Diagrama de actividad:

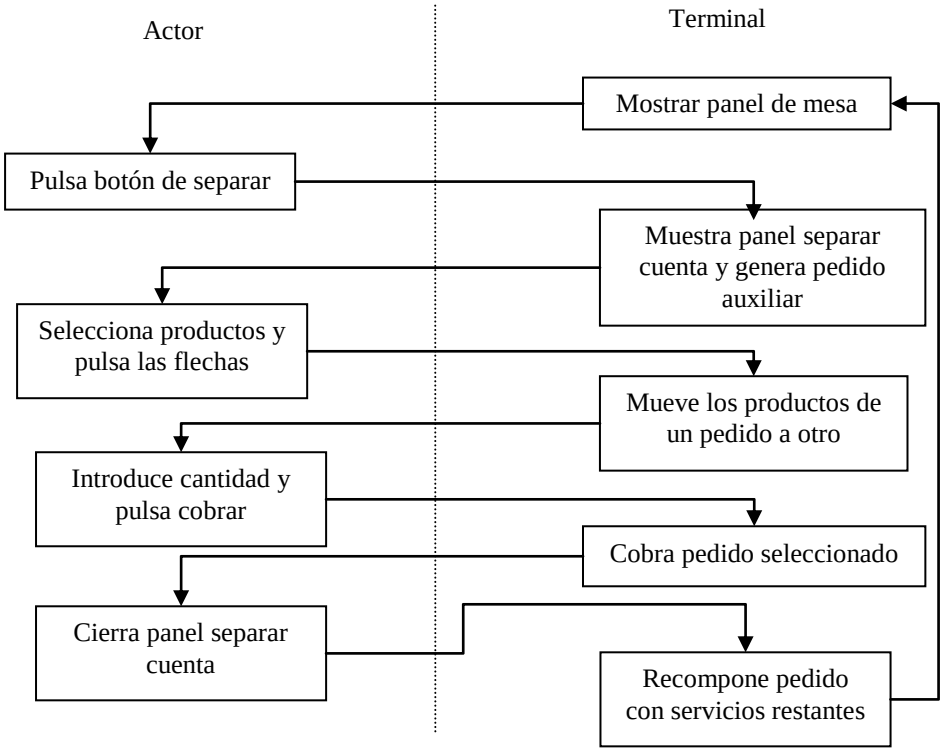


Diagrama 10: Separar cuenta

8. Cobrar.

Objetivo: Cobrar los pedidos.

Actores: Empleados y administradores.

Situación inicial: Panel de visualización de mesas con los datos referentes al pedido actual cargados.

Pasos:

Actor: Introduce una cifra y pulsa el botón de cobrar.

Terminal: Muestra un panel con información sobre el dinero entregado y el cambio del cliente y abre el cajón de dinero. En caso de estar activado, genera un ticket de cobro.

Terminal: Establece la hora de cierre del pedido y lo inserta en la tabla de cobros.

Terminal: Genera un nuevo pedido vacío.

Extensiones:

I.

Terminal: En caso de que el importe total sea mayor que la cantidad entregada muestra un mensaje de error y vuelve al panel de visualizar mesas.

Diagrama de actividad:

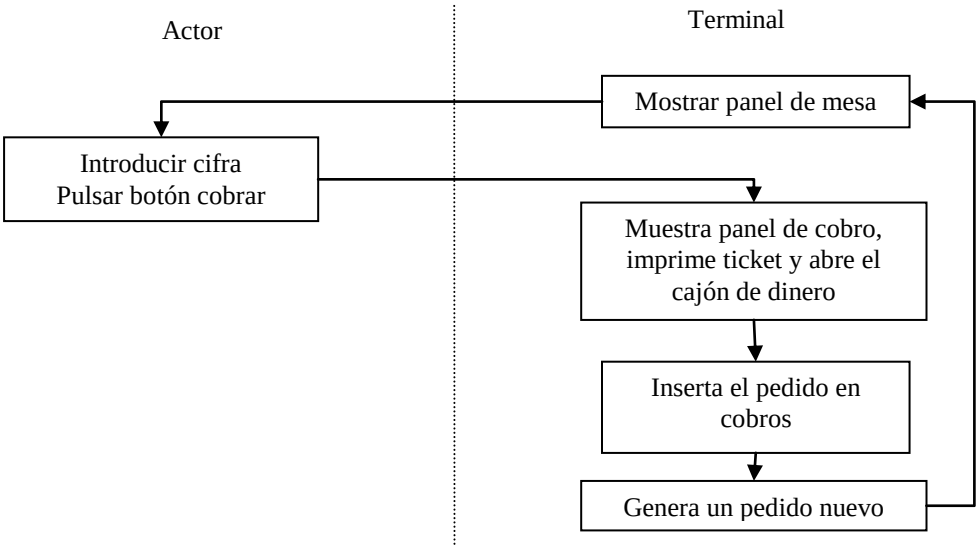


Diagrama 11: Cobrar

9. Cancelar cobro.

Objetivo: Cancelar los cobros que se hayan realizado por error o con errores.

Actores: Empleados y administradores.

Situación inicial: Panel de visualización de mesas con los datos referentes al pedido actual cargados.

Pasos:

Actor: Pulsa el botón cancelar cobro.

Terminal: Muestra el panel de cancelar cobro.

Actor: Introduce un número de pedido o pulsa el botón de último cobro para cargar el número de pedido correspondiente al último pedido cobrado.

Actor: Pulsa el botón cancelar cobro.

Terminal: Genera un nuevo pedido copiando la información del pedido cuyo cobro se va a cancelar. Se introduce dicho cobro en la tabla de cobros cancelados.

Terminal: Asigna el nuevo pedido a la primera mesa de la lista dinámica de mesas disponible.

Extensiones:

I.

Terminal: En caso de que el pedido no exista o no se haya cobrado se generará un mensaje de error.

Diagrama de actividad:

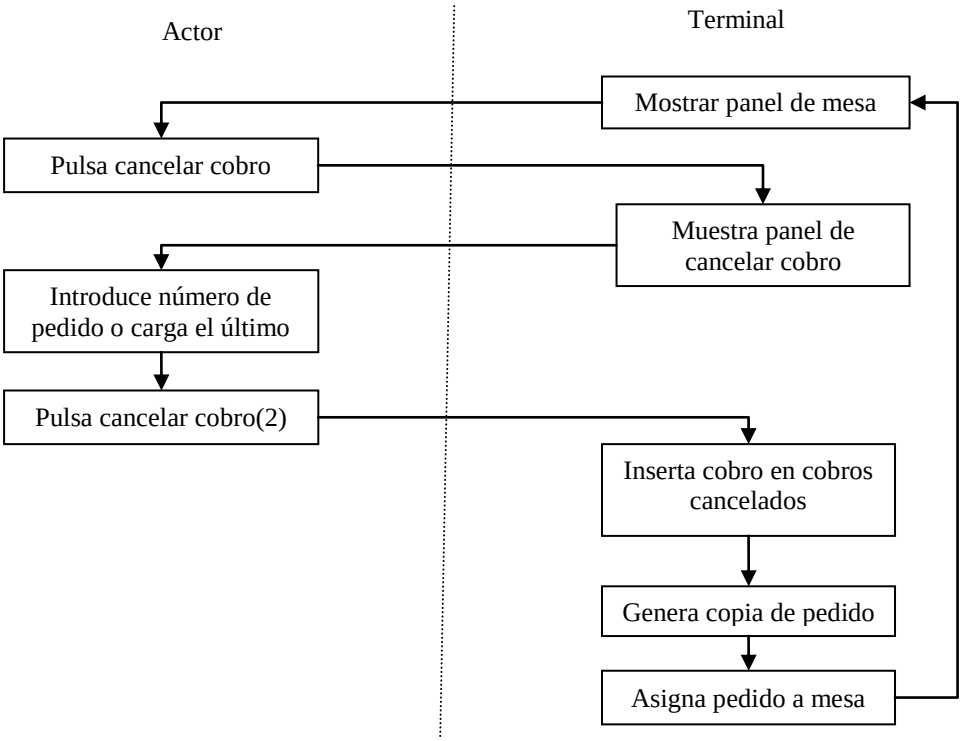


Diagrama 12: Cancelar cobro

10. Abrir jornada de empleado.

Objetivo: Dar de alta al empleado en la jornada activa para registrar las horas de trabajo.

Actores: Empleados y administradores.

Situación inicial: Panel de opciones visible sobre el panel principal.

Pasos:

Actor: Pulsa el botón de ingresar camarero o dependienta.

Terminal: Muestra el panel de añadir camarero o dependienta, según corresponda.

Actor: Pulsa el botón correspondiente a cualquier camarero disponible.

Terminal: Añade el empleado a la lista de jornadas, especificando la hora de entrada del empleado a su puesto de trabajo.

Terminal: Muestra el panel de opciones.

Diagrama de actividad:

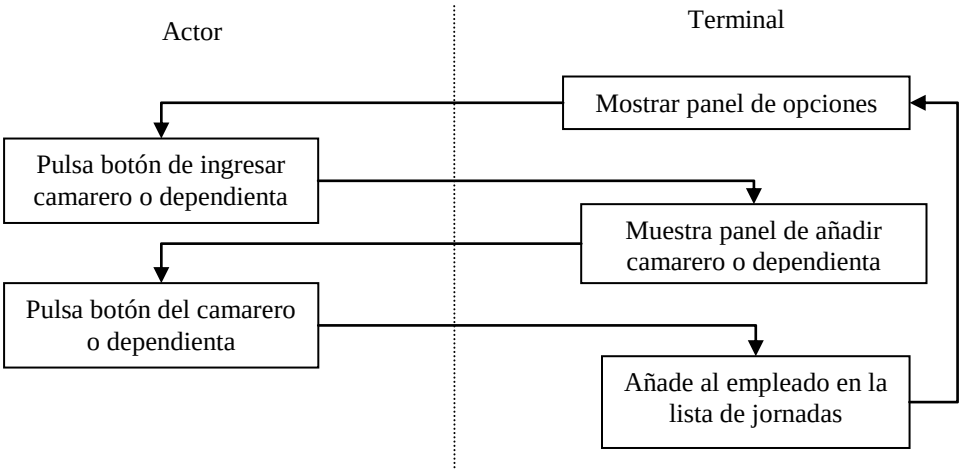


Diagrama 13: Abrir jornada de empleado

11. Cerrar jornada de empleado.

Objetivo: Cerrar la jornada de trabajo del trabajador.
Actores: Empleados y administradores.
Situación inicial: Panel de opciones visible sobre el panel principal.
Pasos:

Actor: Pulsa el botón de sacar camarero o dependienta.
Terminal: Muestra un panel con las jornadas de camareros o dependientas activas.
Actor: Pulsa el botón correspondiente a cualquier camarero que se quiera sacar.
Terminal: Cierra la jornada de trabajo del empleado seleccionado.
Terminal: Muestra el panel de opciones.

Diagrama de actividad:

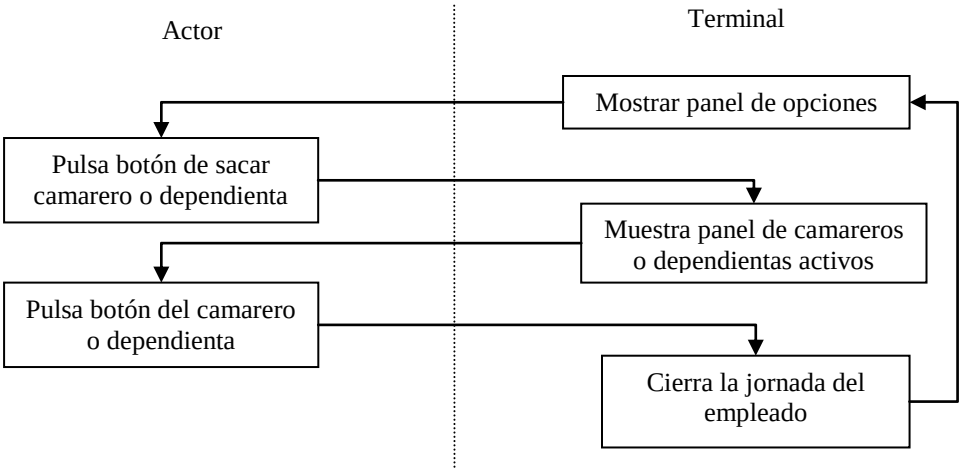


Diagrama 14: Cerrar jornada de empleado

3.1.3 Especificación de las clases

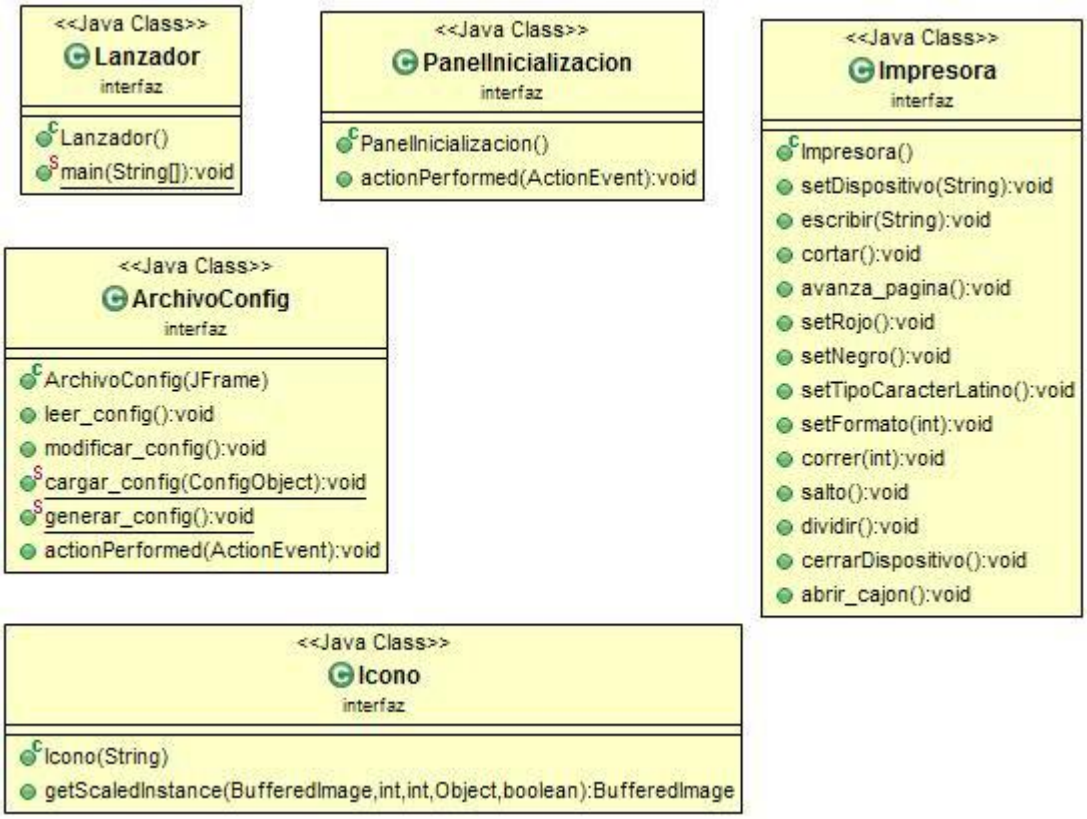


Diagrama 16: Clases independientes

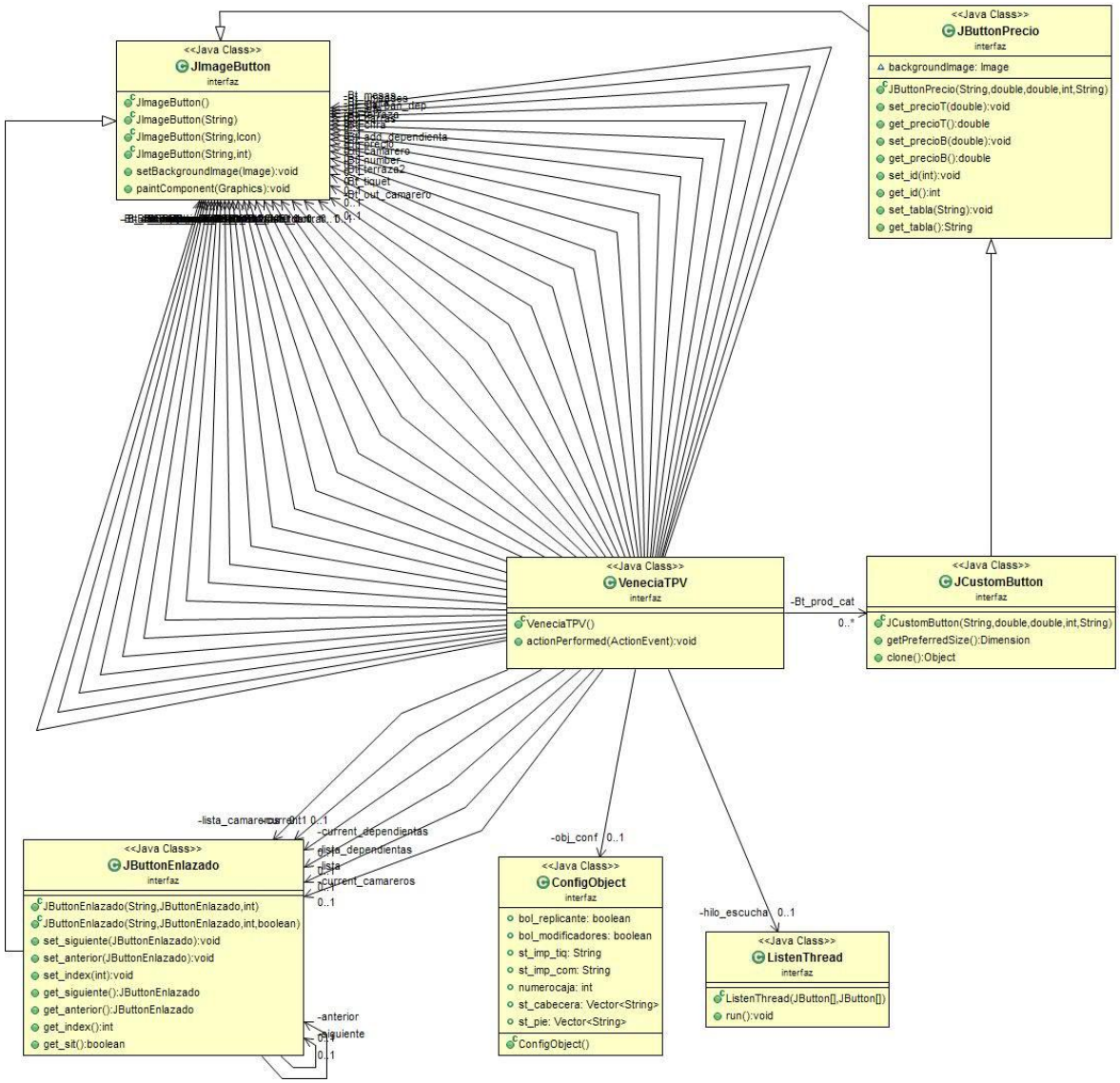


Diagrama 17: Clases relacionadas

Clase Lanzador	
La finalidad de esta clase es simplemente lanzar la aplicación.	
main	Lanza un panel de iniciación

Clase PanelIniciación	
Esta clase muestra el panel de inicialización, el cual permite realizar una configuración o lanzar la aplicación final.	
PanelIniciación	Constructor que instancia todos los componentes del panel de iniciación.
ActionPerformed	Método que gestiona los eventos ocurridos en el panel de iniciación, tales como pulsaciones de botones.

Clase ArchivoConfig	
Esta clase crea un formulario mediante el cual realizar modificaciones en el archivo de configuración del terminal.	
ArchivoConfig	Método constructor cuyo argumento de entrada es el JFrame que llama a este método. Crea la interfaz que servirá para configurar el terminal.
Leer_config	Este método carga la configuración actual en la interfaz de configuración.
Modificar_config	Mediante este método se escriben los datos de configuración en el archivo correspondiente.
Cargar_config	Método para cargar en el objeto ConfigObject los parámetros de configuración actuales. Se trata de un método estático.
Generar_config	Método estático que genera un nuevo archivo de configuración por defecto.

Clase Icono	
Clase utilizada para obtener imágenes redimensionadas respecto a la imagen que se pasa como argumento.	
getScaledInstance	Redimensiona la imagen de entrada en función del ancho y alto objetivo, empleando el método indicado como argumento, con la posibilidad de seleccionar la calidad final.

Clase Impresora	
Clase que permite instanciar una impresora para poder generar tiquets. Esta clase se ha testeado estableciendo un archivo como dispositivo. Falta testearla con una impresora real como se explica más adelante.	
Impresora	Instanciación de objeto impresora.
setDispositivo	Establece el dispositivo que se utilizará como impresora a través de su ruta o etiqueta de volumen de unidad.
Escribir	Envía al dispositivo seleccionado el string a imprimir.
Cortar	Envía a la impresora el comando de corte de papel.
Correr	Genera tantas líneas en blanco como se le indique como parámetro de entrada.
Salto	Realiza un salto de línea.
Dividir	Imprime una línea de guiones con toda la anchura del papel.
cerrarDispositivo	Cierra el dispositivo que estuviese siendo utilizado.
Abrir_cajon	Envía el comando de apertura del cajón de dinero.

Clase JButtonEnlazado	
Mediante esta clase se implementa una lista enlazada de botones. Los cuales se corresponderán a los botones de pedidos auxiliares. Hereda de JImageButton pero añade referencia al botón siguiente y anterior.	
JButtonEnlazado	Método constructor que recibe como parámetros de entrada

	la cadena de texto del botón, una referencia al anterior de la lista y el índice. Además, mediante sobrecarga existe la opción de añadir el tipo de pedido.
set_siguiente	Permite establecer el botón siguiente de la lista.
set_anterior	Permite establecer el botón anterior de la lista.
set_index	Permite establecer el índice del botón.
get_siguiente	Devuelve el botón siguiente de la lista.
get_anterior	Devuelve el botón anterior de la lista.
get_index	Devuelve el índice del botón.
get_sit	Devuelve "true" en caso de tratarse de un pedido de terraza.

Clase JButtonPrecio	
Esta es la clase que implementa los botones del grid de productos. Dichos botones contienen los datos necesarios del producto en cuestión.	
JButtonPrecio	Se trata del constructor, cuyos parámetros de entrada son: el texto del botón, el precio de terraza y barra, el id del producto y la tabla de la base de datos en la que se encuentra.
set_precioT	Permite establecer el precio de terraza del producto.
set_precioB	Permite establecer el precio de barra del producto.
set_id	Permite establecer el id del producto.
set_tabla	Permite establecer la tabla en la que se encuentra el producto.
get_precioT	Devuelve el precio de terraza del producto.
get_precioB	Devuelve el precio de barra del producto.
get_id	Devuelve el id del producto.
get_tabla	Devuelve la tabla a la que pertenece el producto.

Clase JCustomButton	
Clase utilizada para dotar de tamaño a los botones de productos. Hereda de JButtonPrecio.	
JCustomButton	Constructor que recibe los mismos argumentos que la clase JButtonPrecio y que únicamente realiza la llamada al constructor de esta clase (de la que hereda) mediante el uso de "super".
getPreferredSize	Mediante esta clase se devuelve un tamaño fijo preferido. Es utilizado por el layout del contenedor.

Clase JImageButton	
Esta clase extiende de JButton y se utiliza para dibujar una imagen de fondo en el botón y poder mostrar texto encima.	
JImageButton	Constructor sobrecargado hasta 4 veces: sin argumentos se construye un botón sin texto; con texto se crea un botón con

	texto; con texto e icono mostrará el icono y sobre él el texto; y por último, se puede indicar texto, icono y si debe presentarse el texto o no.
setBackgroundImage	Permite asignar la imagen a mostrar como fondo del botón.
paintComponent	Este método dibuja el componente. Es llamado de forma automática y como argumento recibe el objeto Grafics que gestiona la visualización de la aplicación.

Clase ListenThread	
Esta clase implementa el hilo de escucha que imprimirá las comandas enviadas por las PDAs. Hereda de la clase Thread, la cual implementa la interfaz Runnable.	
ListenThread	Constructor que recibe como parámetros de entrada los botones del grid de mesas y barra.
run	Este método arranca el hilo.

Clase VeneciaTPV	
La clase *VeneciaTPV* es la clase principal de la aplicación. Hereda de *JFrame* y constituye la interfaz gráfica principal de la aplicación.	
VeneciaTPV	Método constructor, en el que se inicializan los principales paneles de la aplicación y sus componentes.
crearinterfaz	Método que se llama únicamente tras inicializar la aplicación y que se encarga de añadir las imágenes a los botones de la aplicación. Tanto de la interfaz como de los productos.
db_connect	Método que sirve para realizar la conexión con la base de datos. Deja preparados tres *statements* que serán tres canales de comunicación con la base de datos.
leer_tablas	Mediante este método se crean el grid de familias de productos y los distintos paneles con los productos de cada familia. Obtiene todos los datos de la base de datos.
buscar_en_lista	Comprueba si una mesa se encuentra ya insertada en la lista enlazada de mesas del panel inferior del panel principal. Devuelve true en caso afirmativo y false en caso negativo.
añadir_en_lista	Añade una nueva mesa a la lista enlazada de mesas del panel inferior del panel principal.
eliminar_de_lista	Elimina una mesa de la lista enlazada de mesas del panel inferior del panel principal.
actualizar_grid	Este método actualiza el grid de mesas y barras, mostrando en cada caso el total del importe de la mesa (caso de ser necesario) y modificando el color de las mesas en función de su estado.
actualizar_lista_prod	Este método se encuentra sobrecargado. Si no se le añaden argumentos actualiza la lista de productos del panel de mesa del pedido actual. En caso de añadirle el número de pedido y una lista (*JList*) actualiza esa lista con los servicios del pedido

	indicado.
actualizar_importe	Este método también se encuentra sobrecargado. En el caso de no añadirle argumentos actualizará el importe del pedido actual. De añadirle un número de pedido como argumento, devuelve el importe total del pedido indicado.
actualizar_dueños	Este método se emplea para cargar en los botones de camarero y dependienta los números de los empleados responsables. En caso de no tener asignada la mesa camarero o dependienta se llama al método seleccionar_camarero con el argumento correspondiente.
actualizar_datos	Este método se llama cada vez que se abre una mesa y sirve para llamar a los distintos métodos que cargan o actualizan los datos del pedido: actualizar_lista_prod, actualizar_importe y actualizar_dueños.
borrar_producto	Decrementa en una unidad o borra (si sólo hay una unidad) el servicio seleccionado de la lista de servicios. En caso de no haber ninguno seleccionado se considera seleccionado el último servicio.
obrar_mesa	Con este método se cobra el pedido abierto e imprime un tique de cambio.
cancelar_cobro	Mediante este método se muestra un pequeño panel con el que poder introducir un número de pedido para cancelar el cobro.
abrir_mesa	Este método recibe como argumento un entero que indica si se trata de una mesa de barra o de terraza. Muestra el panel de mesa y carga los datos referentes al pedido.
cerrar_mesa	Método que muestra el grid principal. Tras mostrar el grid principal se llama al método actualizar_grid.
menú_opciones	Este método muestra el panel de opciones sobre el panel principal.
añadir_camarero	Este método crea un JDialog que en el que se muestra a los empleados en el sistema y permite asignarles una jornada activa.
nvitar	Mediante este método el pedido actual se añade a la tabla de invitaciones y se marca como cerrado.
generar_pedido	Este método se emplea para generar nuevos pedidos mientras se mantiene la vista de mesa. Se utiliza tras cobrar una mesa, por si se quiere realizar algún nuevo pedido inmediatamente después, sin salir al panel principal; o cuando se separa cuenta, en cuyo caso el pedido auxiliar que se utiliza se ha generado previamente utilizando este método.
establecer_cantidad	Sirve para fijar una cantidad sobre el servicio de la lista de servicios de mesa seleccionado. Como argumento se le indica la cantidad de servicios que se fijarán.
modif_precio	Método que añade el servicio seleccionado a la tabla de

	modificaciones de precio y le asigna el nuevo precio introducido.
seleccionar_camarero	A este método como argumento se le pasa un entero que indicará si se trata de camarero o dependienta. El método mostrará un panel donde seleccionar a camareros o dependientas con jornada activa, o bien para finalizar su jornada actual de trabajo o bien para asignarlos como responsables de la mesa.
asignar_a_mesa	Mediante este método se modifica el pedido en la base de datos especificando el camarero o dependienta responsable.
imprimir	Mediante este método se instancia un objeto de la clase impresora y se genera un tique de pedido que será impreso por la impresora especificada en la configuración. Este método se encuentra también sobrecargado. En caso de añadirle como argumentos el dinero de entrega y el cambio, imprime un tique de cobro, rellenando dichos datos del tique con los datos introducidos.
menú_configuración	Este método, tras llamar al método de introducir contraseña y autenticar la contraseña introducida, en caso de ser positivo el resultado instancia un ArchivoConfig.
menú_jefe	Este método, tras pedir introducción de contraseña y validarla, crea el panel donde se pueden gestionar los movimientos de caja (ingresos o retiradas).
comprobar_pass	Utiliza el método encryptar_sha1 para codificar el String que se le pasa como argumento y comprueba si existe en la base de datos alguna entrada que coincida en la tabla de permisos.
encryptar_sha1	Este método hace uso de la clase MessageDigest para encriptar el String que se le pasa como argumento mediante el algoritmo SHA-1 y devuelve un String con el hash generado.
introd_pass	Genera el panel de introducción de contraseña y gestiona todos los eventos que en el ocurren (pulsar las teclas y demás).
juntar_mesas	Este método establece el modo juntar y cierra la mesa. La próxima vez que se abra una mesa se juntará el pedido indicado en la variable pedido_juntar.
separar_cuenta	Este método genera el panel de separación de cuentas. Permite gestionar los servicios entre el pedido actual y uno auxiliar que se genera. Mediante la utilización de las flechas se pueden mover los servicios entre los pedidos. Y con los botones de cobrar se pueden cobrar cualquiera de los dos pedidos resultantes.
res_cifra	Mediante este método se realiza un reset del botón que indica la cifra introducida. De esta manera, cada vez que se utiliza se resetea para no mantener el valor para futuros usos.

3.1.4 Relación entre casos de uso y clases.

1. Configuración de la aplicación	ArchivoConfig; ConfigObject
2. Movimientos de caja	VeneciaTPV
3. Cerrar aplicación	VeneciaTPV
4. Abrir editar o generar pedido actual	VeneciaTPV
5. Modificar productos de pedido actual	VeneciaTPV
6. Modificar empleados de pedido	VeneciaTPV
7. Imprimir ticket	VeneciaTPV;Impresora
8. Invitar	VeneciaTPV
9. Juntar pedido	VeneciaTPV
10. Cobrar	VeneciaTPV
11. Cancelar cobro	VeneciaTPV
12. Abrir jornada de empleado	VeneciaTPV
13. Cerrar jornada de empleado	VeneciaTPV

3.2. Diseño

Durante la fase de diseño, se han definido las estructuras principales de la base de datos, los casos de uso y la interfaz de usuario.

3.2.1 Diseño de persistencia

Debido a la necesidad de poder realizar consultas futuras sobre las ventas realizadas, incluso en diferentes jornadas de facturación, para poder llevar la contabilidad del negocio así como obtener estadísticas varias (productos, sabores o tamaños más vendidos, por ejemplo), es necesario que el sistema de almacenamiento de información sea persistente.
Es por esto, por lo que se recurre al uso de una base de datos relacional para poder mantener una cantidad masiva de datos bien organizados para poder hacer recuperaciones de manera eficiente.

Para poder realizar el diseño de la base de datos lo primero que se debe definir perfectamente es "qué" información se debe guardar. Una vez que se han identificado los datos que deben guardarse se pasará a diseñar el "cómo" guardarlos.

3.2.1.1 Requisitos

La información, es la herramienta más poderosa en el mundo empresarial. Permite tomar decisiones de negocio con mayor precisión y menor riesgo. Sin embargo, una saturación de información puede provocar el efecto contrario. Por lo tanto, es necesario identificar claramente los datos relevantes a tener en cuenta para diseñar la base de datos.

La información que se debe almacenar para este proyecto es la siguiente:

- Empleados.
  La base de datos contendrá una lista con los empleados, así como el departamento al que pertenecen.
  Se deberá controlar la hora de entrada y salida en las jornadas de trabajo, así como el trabajo realizado en forma de pedidos servidos.

- Familias de productos.
  Debe guardarse el nombre de cada familia de productos, con el fin de identificar claramente la familia a la que pertenece cada producto.
  Esta información puede ser modificada en cualquier momento por el administrador o gerente de local permitiendo añadir, editar o eliminar familias de productos.

- Productos.
  La relación de productos, con sus correspondientes descripciones, precios y familia a la que pertenecen, son otro elemento a incorporar en la base de datos.
  Esta información también podrá ser añadida, modificada o eliminada en cualquier momento por parte de un administrador.

- Pedidos.
  Los continuos pedidos que se vayan generando también se almacenarán para poder tener un histórico de ventas así como gestionar los pedidos abiertos. Al tratarse de un local con servicio de terraza, van a haber varios pedidos abiertos que pueden ser modificados en cualquier momento.

- Servicios.
  Para poder saber qué cantidad de cada producto se ha vendido y con ello, poder obtener estadísticas que permitan tomar decisiones acerca de los productos que se ofertan, es necesario registrar cada servicio que se hace.

- Modificaciones.
  Se almacenarán también las modificaciones que pueda sufrir cualquier servicio (sabor, ingredientes excluidos o extras, etc).

- Cobros.
  Guardar cada cobro que se hace, con la información sobre el dinero que entregan los clientes y la hora y sesión de caja en la que se realiza el cobro permitirá calcular la caja diaria y resolver posibles problemas con errores en la introducción del dinero para el cobro. También se utilizará para indicar en cuál de las cajas del establecimiento se ha realizado el cobro.

- Movimientos de caja.
  Se registrarán también los posibles movimientos en forma de ingresos (el cambio inicial de la caja o las posibles incorporaciones de efectivo que pudiera ser necesario) o de retiradas (en caso de exceso de dinero en caja puede ser necesario extraer dinero a lo largo de la jornada de trabajo).

- Invitaciones.
  También se almacenarán los sucesivos pedidos referidos a invitaciones.

- Borrados y modificaciones especiales.
  Con el fin de reflejar todo lo que ocurre con los pedidos, desde que son generados hasta que finalmente se cobran, se registrarán todos los cambios que ocurran en dicho pedido, incluidos los servicios borrados o cambios de precios de carácter especial.
  Este es un requisito exigido explícitamente por el cliente de la aplicación, con el fin de poder controlar los movimientos que realizan los empleados sobre los pedidos. Pudiendo comprobar con ello si se producen operaciones que puedan sugerir alguna actividad desleal por parte de algún empleado (reducir el precio de los servicios en caja para sustraer el sobrante, borrar pedidos completos en lugar de cobrarlos en caja, etc).

3.2.1.2 Diseño relacional

Ante los citados requisitos se ha diseñado la estructura de la base de datos como se puede observar en la figura 12. A continuación se analizarán las tablas utilizadas.

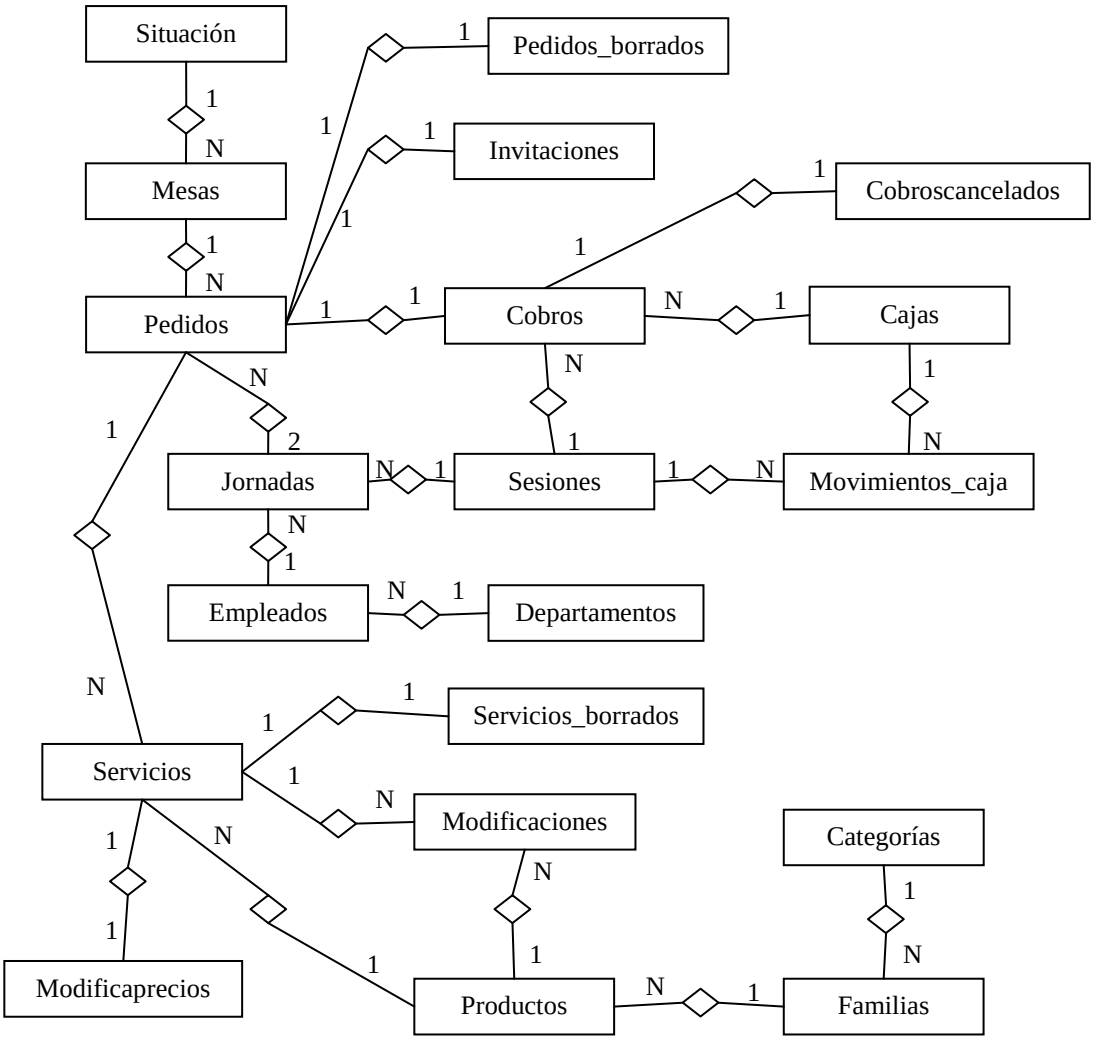


Diagrama 15: Diagrama entidad-relación de la base de datos

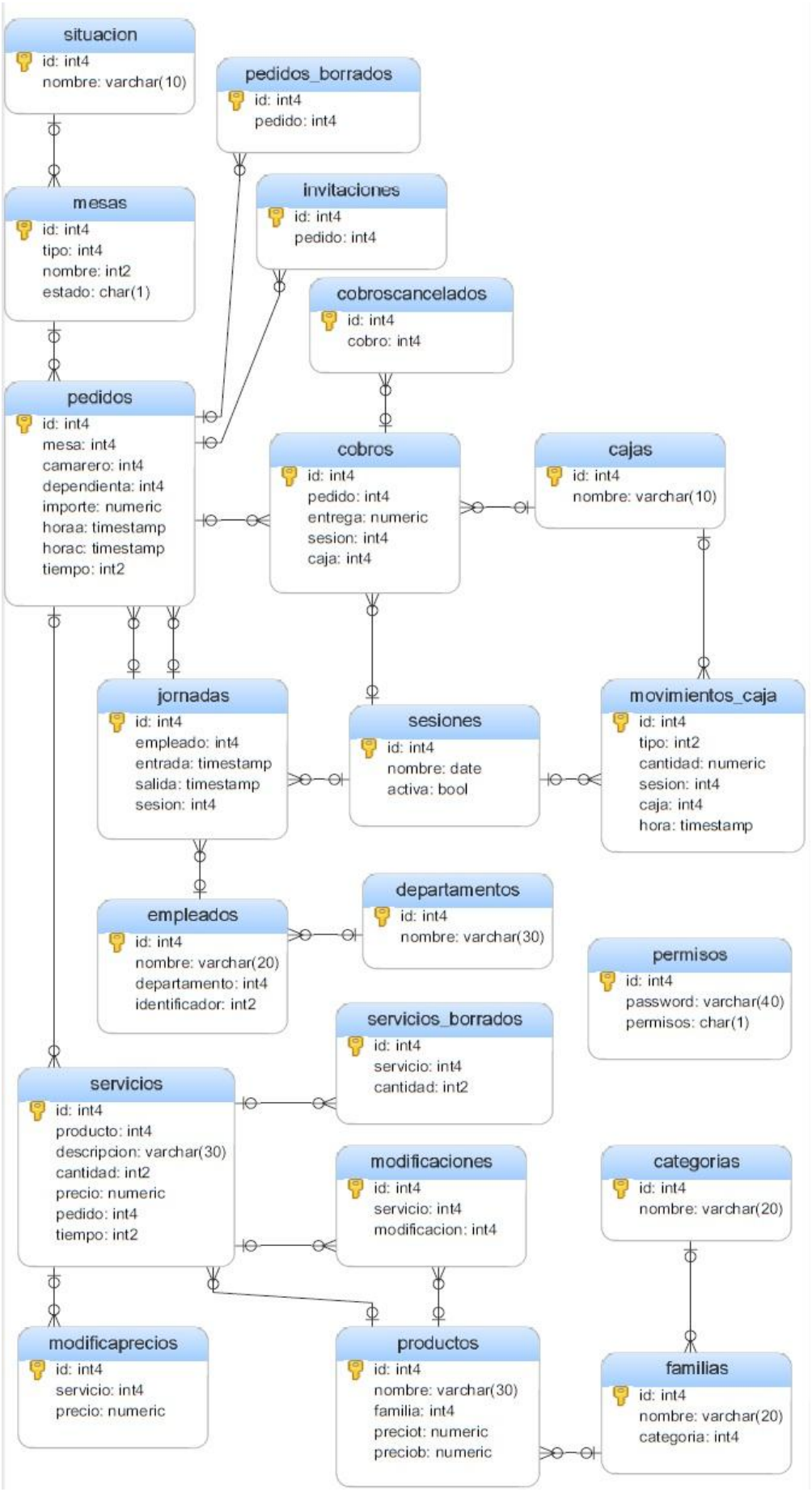


Figura 12: Estructura de la base de datos extraída con Navicat Data Moduler

Como se puede observar, a todas las tablas se les ha añadido un campo de indexación con el nombre índice.

3.2.1.3 Análisis de las tablas:

- **Cajas:**
  Esta tabla se emplea para registrar las distintas cajas que hay en el establecimiento. Para el caso de este proyecto sólo habrá dos cajas en cada establecimiento.

ID	Nombre

- **Categorías**:
  Mediante el uso de esta tabla se distinguirán dos categorías diferentes de artículos (o más si los hubiera): productos y modificadores.

ID	Nombre

- **Cobros:**
  Esta tabla reflejará las entradas de dinero a la caja derivadas del cobro de los pedidos.
  El campo pedido indica el pedido al que pertenece este cobro.
  Mediante el campo entrega se registra el dinero que entregó el cliente para realizar el pago.
  El campo sesión indica la sesión de caja en la que se encuadra el pedido. Las sesiones representarán las jornadas de trabajo.
  El campo caja indica en qué caja se realizó el cobro.

ID	Pedido	Entrega	Sesión	Caja

CF: Pedidos(id)

- **Cobroscancelados:**
  Esta tabla registrará los cobros que han sido cancelados.

ID	Cobro

- **Departamentos:**
  En esta tabla se encontrarán los distintos departamentos que a los que pueden pertenecer los empleados. Por el momento se distinguirán: dependientas y camareros.

ID	Nombre

- **Empleados:**
  Ofrece un listado de los empleados, indicando el departamento al que pertenecen y un número identificador propio.

ID	Nombre	Departamento	Identificador

CF: Departamentos(id)

- **Familias:**
  Las familias son las sub-categorías en las que se pueden agrupar los artículos. Cada familia pertenece a una categoría.

ID	Nombre	Categoría

CF: Categorías (id)

- **Invitaciones:**
  En esta tabla se recogerán todos los pedidos correspondientes a invitaciones por parte de la empresa.

ID	Pedido

CF: Pedidos(id)

- **Jornadas:**
  Las jornadas constituyen porciones de tiempo en las que un trabajador se encuentra en su puesto de trabajo. Permitirá obtener las horas de trabajo de cada trabajador para calcular el salario.
  Cada entrada constará de un campo de identificación del empleado, otros campos con la hora de entrada y de salida respectivamente, y la sesión de caja a la que pertenece la jornada.

ID	Empleado	Entrada	Salida	Sesión

CF: Empleados(id) CF: Sesiones(id)

- **Mesas:**
  Supone un listado de todas las mesas existentes, se le indica su situación, nombre y estado. Dicho estado puede ser:
  - Abierto (a): Este estado significa que la mesa tiene asociado un pedido que aún no ha sido cobrado.
  - Cerrado (c): En este estado la mesa se declara como "vacía", no hay clientes en ella y todos sus pedidos asociados se encuentran cobrados.
  - Uso (u): Este estado indica que otro usuario, desde otro terminal, está accediendo a los datos de la mesa.

ID	Tipo	Nombre	Estado

CF: Situación(id)

- **Modificaciones:**
  En esta tabla se indican las modificaciones que cualquier producto pueda sufrir. Dichas modificaciones serán referencias a la tabla de productos donde se referirá a un producto de alguna sub-categoría perteneciente a la categoría de modificaciones.

ID	Servicio	Modificación

CF: Servicios(id) CF: Productos(id)

- **Modificaprecios:**

La modificación de los precios de los productos está considerada modificación especial y por tanto se recoge expresamente en esta tabla, la cual indica el servicio y el precio al que se establece.

ID	Servicio	Precio

CF: Servicios(id)

- **Movimientos_caja:**
  En esta tabla se recogerán los movimientos de ingreso o extracción de efectivo de la caja por tareas administrativas. Se recogerán los datos de: tipo extracción o ingreso, la cantidad, la sesión de caja, la caja donde se realiza y la hora a la que se realiza.

ID	Tipo	Cantidad	Sesión	Caja	Hora

CF: Sesiones(id) CF: Cajas(id)

- **Pedidos:**
  Esta tabla acumulará todos los pedidos que se vayan sucediendo. A cada pedido se le asignará:
  - Un identificador único (Id).
  - Un identificador de la mesa en la que se sitúa el pedido.
  - Identificadores de camarero y dependienta responsables de atender la mesa.
  - El importe actualizado del pedido.
  - Las horas a las que se abre y cierra la mesa. Las mesas no cobradas permanecerán con la hora de cierre a "null".
  - El tiempo de mesa. Este tiempo indicará el número de veces que se ha consultado la mesa. A cada nueva consulta, con o sin modificaciones, este valor se incrementará, permitiendo con ello identificar temporalmente varias rondas de servicios con el fin de facilitar al cliente la lectura de la factura, no acumulando los servicios que, aún haciendo referencia al mismo producto, se hayan realizado en distintas ocasiones.

ID	Mesa	Camarero	Dependienta	Importe	HoraA	HoraC	Tiempo

CF: Mesas(id), Jornadas(id), Jornadas(id)

- **Pedidos_borrados:**
  Permitirá listar los pedidos borrados, para poder evitar repercutirlos en el resumen económico de la sesión de caja.

ID	Pedido

CF: Pedidos(id)

- **Permisos:**
  Esta es la única tabla sin relaciones. Se emplea para almacenar las distintas contraseñas que permiten acceder a las partes administrativas de la aplicación. Todo el potencial de esta tabla se extraerá en la aplicación de

gestión que será objeto de desarrollo en líneas futuras y por tanto no entra dentro del alcance de esta aplicación (como se explica en las motivaciones y objetivos).
La tabla consta de un campo con la contraseña encriptada mediante encriptación SHA-1 (como se verá más adelante) y una identificación del tipo de permisos que se obtienen al introducir esa contraseña. Sin embargo esta última parte no se empleará en este proyecto.

ID	Password	Permisos

- **Productos:**
  Esta tabla contendrá los productos disponibles para vender o servir. Está compuesta de un campo con el nombre del producto en cuestión. A continuación se indica la familia a la que pertenece dicho producto. Los campos "preciot" y "preciob" indican el precio del producto, en terraza y en barra, respectivamente.

ID	Nombre	Familia	PrecioT	PrecioB

CF: Familias(id)

- **Servicios:**
  En esta tabla se irán acumulando los servicios correspondientes a cada pedido.
  El campo producto hace referencia al producto en concreto que se ha seleccionado.
  La descripción permite dar una descripción modificable para cada servicio en concreto.
  Mediante el campo cantidad se puede seleccionar la cantidad de productos semejantes que están asociados.
  A continuación se indican el precio y el pedido al que pertenece el servicio.
  Y para finalizar el tiempo de mesa en la que el servicio fue asociado al pedido.

ID	Producto	Descripción	Cantidad	Precio	Pedido	Tiempo

CF:Productos(id) CF: Pedidos(id)

- **Servicios_borrados:**
  En esta tabla se reflejarán las cantidades de servicios que se borran. Si las cantidades de servicios y las cantidades de servicios borrados son iguales, el servicio habrá sido borrado totalmente del pedido.

ID	Servicio	Cantidad

CF: Servicios(id)

- **Sesiones:**
  Esta tabla registra las sesiones de caja. El nombre indica el día en el que comienza la sesión, mientras que el campo "activa" indica si la sesión se encuentra activa actualmente.

ID	Nombre	Activa

- **ituación:**
  Esta tabla contendrá las distintas localizaciones de las mesas. Las mesas serán los lugares donde potencialmente se situará a los diferentes clientes. A priori existen mesas de barra y de terraza.

ID	Nombre

3.2.2 Interfaz gráfica de usuario

El diseño de la interfaz gráfica de usuario se ha realizado en base de la distribución de interfaz de usuario de la aplicación que actualmente posee el cliente. De esta manera, se reducirá drásticamente la curva de aprendizaje en la utilización de la nueva aplicación, además de cumplir un requerimiento explícito del cliente.
Partiendo de la aplicación antigua y obsoleta conocida por el cliente, se han añadido las nuevas funcionalidades en función de los requerimientos, así como se han eliminado elementos desusados y se ha redistribuido la funcionalidad de una manera más lógica, eficiente e intuitiva.
La interfaz de usuario se ha diseñado como sigue:

3.2.2.1 Panel de inicialización o lanzador.

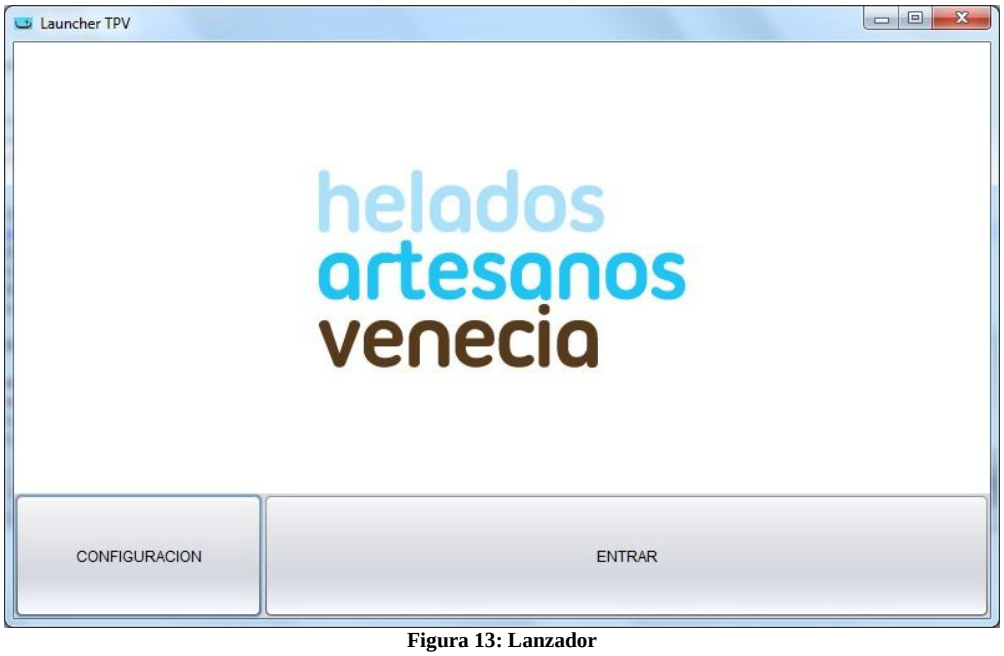


Figura 13: Lanzador

Es el panel que se muestra al iniciar la aplicación. Este panel permite realizar una configuración previa al arranque del terminal y lanzar la aplicación principal.

3.2.2.2 Panel principal.

El panel principal muestra el “grid” de mesas, en el que se pueden visualizar los pedidos abiertos o en uso.

Figura 14: Panel principal

Se puede observar la señalización especial que tiene cada mesa en función de su estado. Mientras que las mesas marcadas en rojo son las mesas que se encuentran en uso por otros usuarios y se encuentran bloqueadas, por lo que al pulsarla se mostrará un mensaje informando de ello. Las mesas en azul más oscuro indican que la mesa tiene un pedido actualmente abierto, y además se indica el importe actual del pedido. Las mesas en color azul claro indican que no están en uso actualmente.
En el panel inferior, como se puede ver, se van añadiendo las mesas que se abren y no están comprendidas entre el 1 y el 49 (en este caso, la mesa 50).
Los botones de barra y terraza permiten alternar la vista de los pedidos entre los correspondientes a barra y terraza respectivamente.
El botón de opciones despliega el menú de opciones.

3.2.2.3 Menú de opciones.

Figura 15: Panel de opciones 1

Como se puede apreciar, el menú de opciones permite comenzar y finalizar las jornadas de los empleados. Permite modificar la configuración y acceder al panel de movimientos de caja. Por último, permite cerrar la aplicación.

3.2.2.4 Configuración.

Figura 16: Configuración

Se puede seleccionar en cuál de las cajas se registrarán los cobros de este equipo y si se encarga de realizar la replicación de la base de datos. También se puede configurar tanto la cabecera como el pie del ticket e indicar la etiqueta de la impresora para los tickets.
Las opciones desactivadas se activarán en futuras versiones.
Mediante el botón "por defecto" se puede crear un nuevo archivo de configuración cargado con unos valores por defecto. De esta manera, en caso de que se editase el archivo de configuración de forma manual generando una versión corrupta no válida o en caso de ser eliminado, se podría generar un nuevo archivo de configuración de acuerdo al formato requerido.

3.2.2.5 Añadir camarero o dependienta.

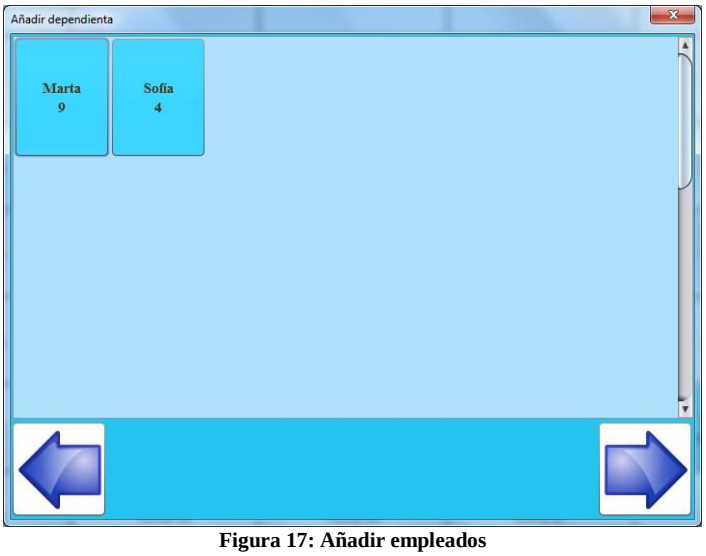


Figura 17: Añadir empleados

Este menú se cargará con los camareros disponibles en caso de pulsar en ingresar camarero. La figura 13 muestra lo que se visualiza tras pulsar en ingresar dependienta.
En este panel, pulsando sobre los empleados se les introduce en la lista de jornadas activas y se convierten en seleccionables para ser responsables de mesas.
Mediante las flechas, se puede navegar entre los distintos paneles de empleados que se generarían en caso de que se superase la capacidad del panel.

3.2.2.6 Seleccionar empleado.

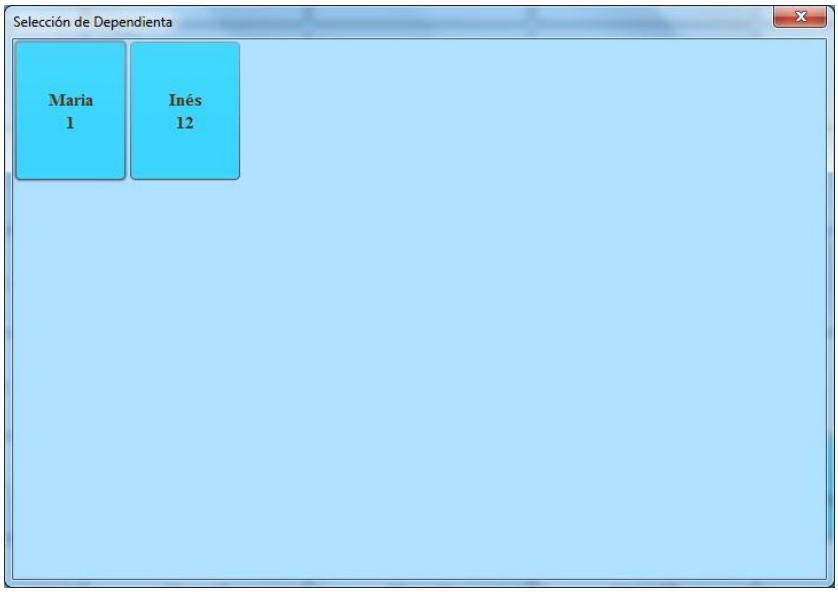


Figura 18: Seleccionar empleado

Este panel se muestra cuando se pulsa en sacar camarero o dependienta y cuando se pulsa sobre los iconos correspondientes al de camarero y camarera respectivamente dentro de la visualización de las mesas (se verá más adelante).
Seleccionando un empleado tras haber accedido pulsando el botón para sacar a algún empleado se cerrará la jornada de dicho empleado.
Seleccionando un empleado tras haber pulsado los botones de camarero o dependienta de la visión de mesas se asignará el empleado seleccionado a la mesa correspondiente.

3.2.2.7 Menú “el jefe”.

Figura 19: El jefe

Mediante este menú se puede ingresar o retirar dinero de la caja y registrarlo en la base de datos.

3.2.2.8 Panel de mesa.

Figura 20: Panel de mesa

Este es el panel más importante a la hora de crear y gestionar las ventas. Se visualizan los servicios de los pedidos, el número identificador de los camareros responsables de servir la mesa y el importe total del pedido, así como ofrece una completa interfaz para manipular dichos datos.
En la parte superior se encuentra la barra de números, que sirve para introducir cifras. A continuación vemos un panel de familias en el que se muestran las familias de productos disponibles.
El sector inferior izquierda contiene los paneles de los productos de cada familia.
A la derecha se puede ver la lista de servicios del pedido así como las demás opciones: selección de empleados, imprimir, volver al grid de mesas, borrar, establecer número de unidades, modificar precio, invitar, separar cuenta, juntar cuenta, cobrar y cancelar cobro.

3.2.2.9 Separar cuenta.

Figura 21: Separar cuenta

Este panel muestra dos listas entre las que se puede distribuir los servicios de un pedido para realizar el cobro por partes. Con el teclado numérico inferior se puede introducir la cantidad entregada por el cliente. Con los botones de cobrar se cobra cada una de las listas de productos. Para distribuir los servicios entre ambas listas se emplean las flechas de izquierda y derecha.

3.2.2.10 Cancelar cobro.

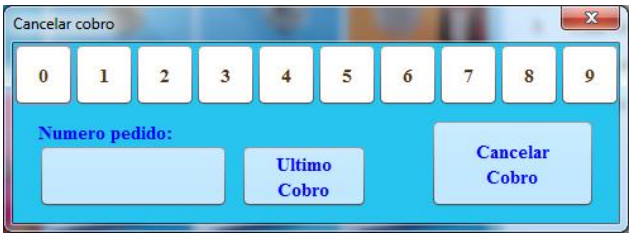


Figura 22: Cancelar cobro

Mediante este panel se puede cancelar el último cobro realizado o el cobro de cualquier pedido introduciendo el número de pedido.
Mediante esta funcionalidad se pueden solucionar errores de cobro que ocurren en ocasiones debido a la alta carga de trabajo.

Se puede observar, que el diseño general de la interfaz es bastante sencillo y gracias a esto el manejo de la aplicación resulta muy intuitivo.
El color adoptado sigue los patrones de los colores de la empresa cliente de la aplicación. Las imágenes empleadas para la representación de los productos se encuentran en un directorio donde se deben incorporar las imágenes que se quieran mostrar. La asociación entre imágenes y productos se realiza por medio del nombre, lo que implica que las imágenes deben nombrarse de igual manera que los productos para poder ser reconocidas y encontrarse bajo la extensión de imagen “.jpg”.

3.3. Implementación.

3.3.1. Modelo Vista Controlador (MVC)

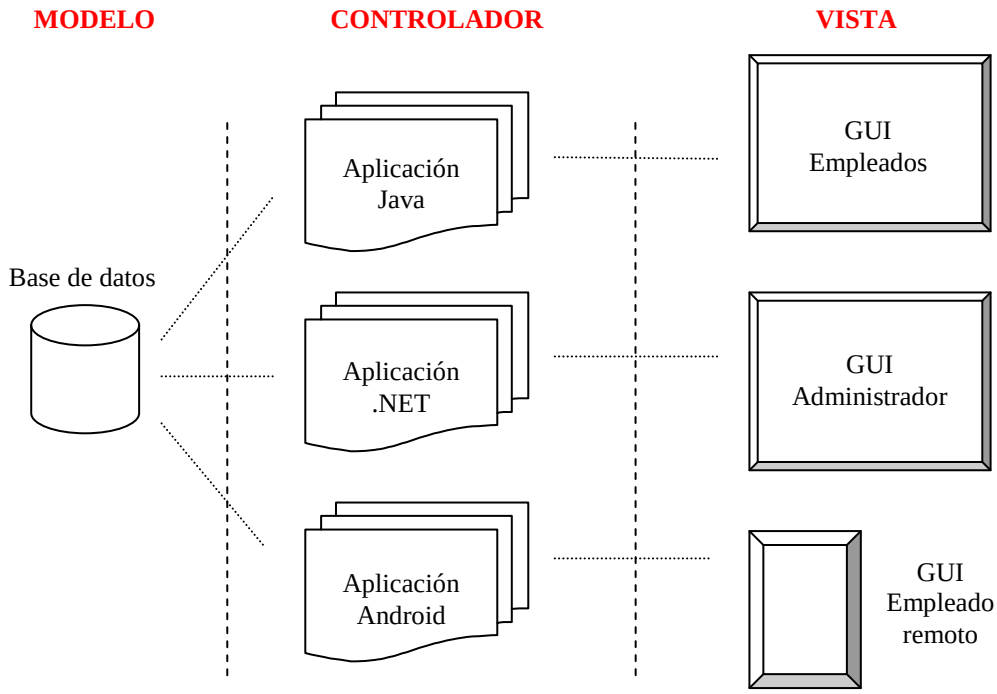


Figura 23. Arquitectura MVC

Como se puede ver en la Figura 23, la estructura de la aplicación está dividida siguiendo el modelo del patrón MVC. Este patrón distingue claramente tres elementos: el modelo, el controlador y la vista.
Las aplicaciones visuales en las que existe una fuerte interactividad son las que más se prestan a este tipo de arquitectura de software.
El modelo es el elemento encargado de gestionar o almacenar los datos sensibles de la aplicación. En el caso de este proyecto la capa de modelo se corresponde con la base de datos.
La vista es la capa software que provee de una interfaz gráfica de usuario para la aplicación. Para este proyecto la vista se corresponde a la interfaz de usuario creada en Java para los empleados.
El controlador es la capa software que enlaza el modelo y la vista, es decir, gestiona el paso de información del usuario a través de la vista al modelo de gestión de información. La aplicación Java es la que se encarga de realizar las funciones de controlador.
Este modelo es ampliamente utilizado por la posibilidad de reutilización de las diferentes partes en las que se compone, así como la posibilidad de modificar alguna de las capas independientemente de las demás. Es decir, se podría modificar la estructura de la interfaz independientemente del modelo, sin afectarlo, y viceversa.

3.3.2. Herramientas

Para el desarrollo de este proyecto se han empleado las herramientas que se describen a continuación.

3.3.2.1 Navicat

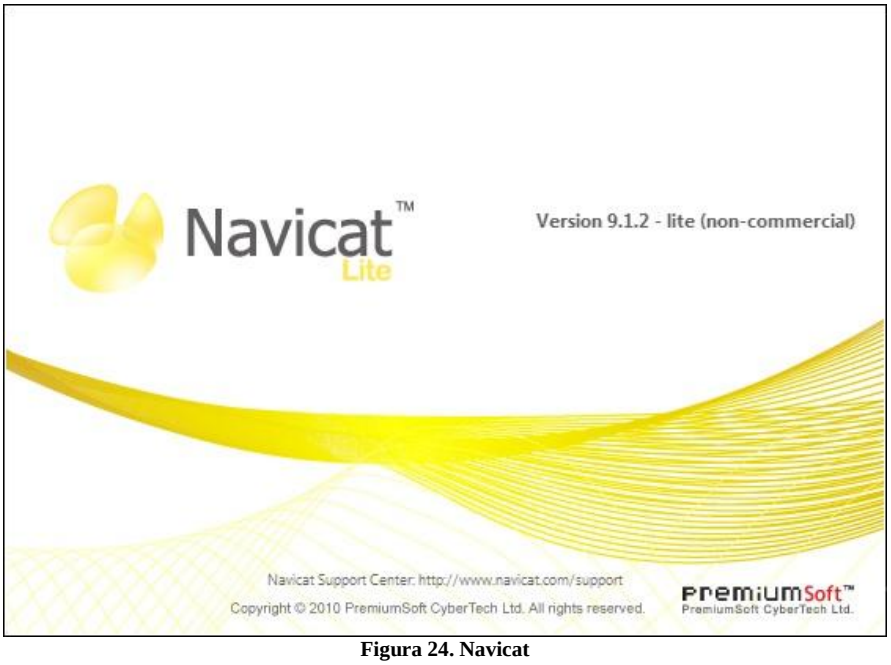


Figura 24. Navicat

Navicat es una herramienta de gestión y administración de bases de datos. Permite la conexión con diversos tipos de bases de datos tales como: MySQL, PostgreSQL, Oracle, SQLite y SQL Server.
Ofrece la posibilidad de gestionar múltiples conexiones simultáneamente a diferentes servidores. Aporta una interfaz gráfica sencilla para la creación y manejo de la estructura de las tablas que conforman la base de datos.

La versión empleada, como se puede ver en la figura anterior, es: Navicat 9.1.2 – lite (no-comercial). En esta versión, algunas de las funcionalidades se encuentran bloqueadas. Esta versión no se encuentra actualmente disponible por parte de una fuente oficial de la empresa desarrolladora de Navicat. Dicha empresa ha modificado recientemente su catálogo de productos retirando este producto y ofreciendo un paquete específico para cada tipo de base de datos y una versión Premium, con toda la funcionalidad, que engloba a todos los tipos de bases de datos.

En la Figura 25 se puede ver el espacio de trabajo de Navicat. Como se puede observar, en la parte superior se encuentra una barra de menús típica y una barra de herramientas que permite gestionar nuevas conexiones con los distintos tipos de bases de datos, los usuarios de cada tipo de base de datos, una serie de vistas como tablas, *views*, funciones, etc, y una serie de opciones bloqueadas, debido a que estamos trabajando con la versión lite.
En la parte central, a la izquierda, se observa un panel que contiene las distintas conexiones a distintos tipos de datos que pueden haber configuradas. En la figura mostrada, se observa como la conexión PostgreSQL-DB está abierta y además vemos como consta de dos bases de datos, las cuales son *postgres* y *Venecia*.
En la parte central, a la derecha, vemos las distintas tablas que contiene el esquema *"public"* de la base de datos PostgreSQL Venecia. El contenido de este panel varía en función de la vista seleccionada en la barra de herramientas.

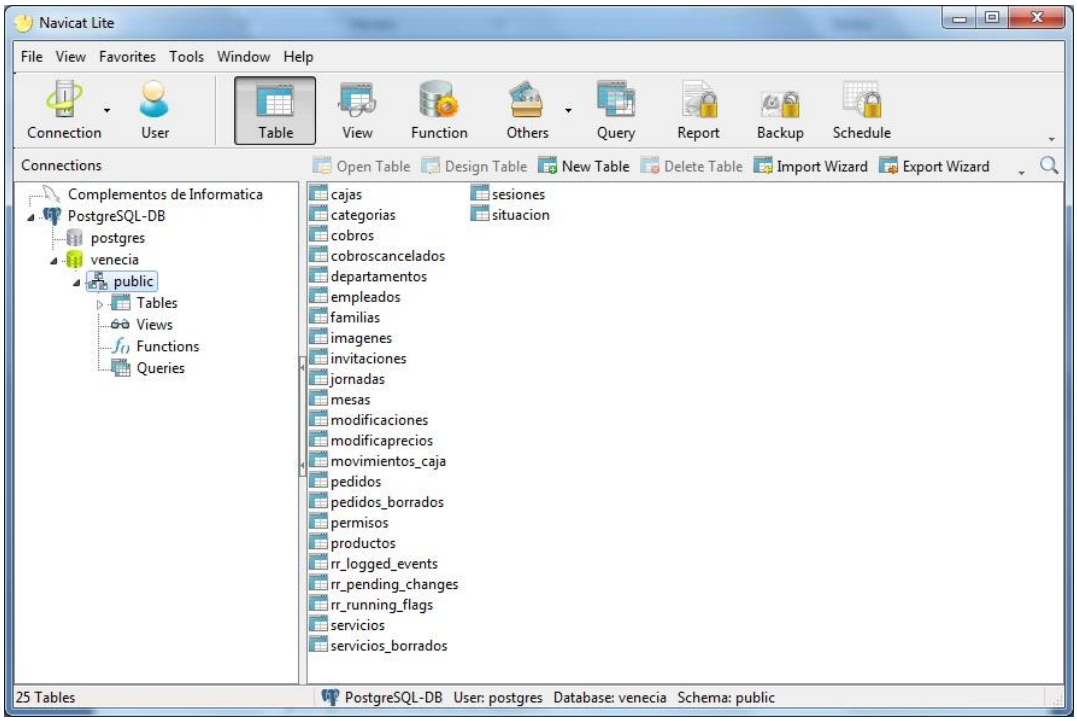


Figura 25. WorkSpace Navicat

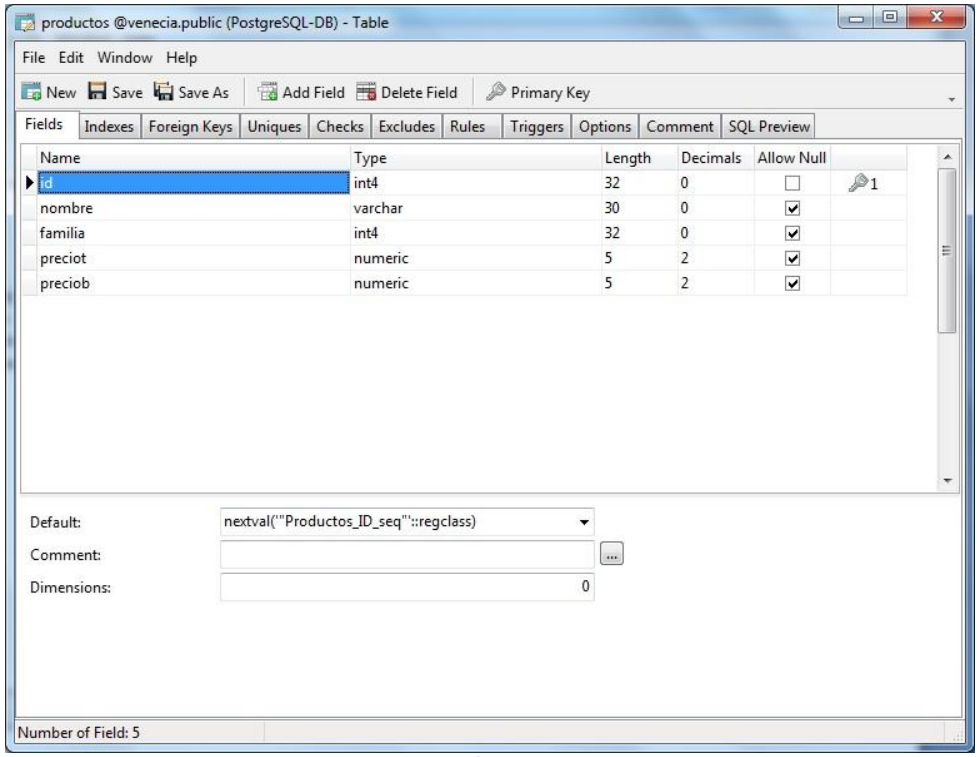


Figura 26. Edición de tablas Navicat

En la ventana de la Figura 26 se observa la interfaz que ofrece navicat para la creación y edición de tablas. Como se puede observar, dota de una interfaz sencilla con la que poder gestionar los campos de las tablas, los tipos de datos, los índices, las claves externas, campos únicos, etc.

3.3.2.2 Navicat Data Modeler

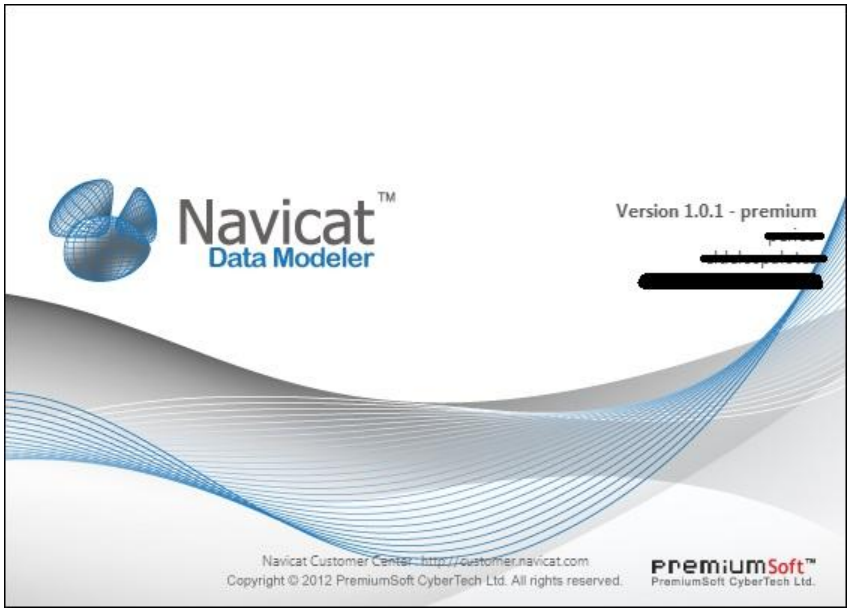


Figura 27: Navicat Data Modeler

Navicat Data Modeler es una herramienta de diseño de bases de datos que sirve para visualizar el esquema de relaciones de bases de datos existentes, o crear bases de datos a partir del esquema de relaciones.

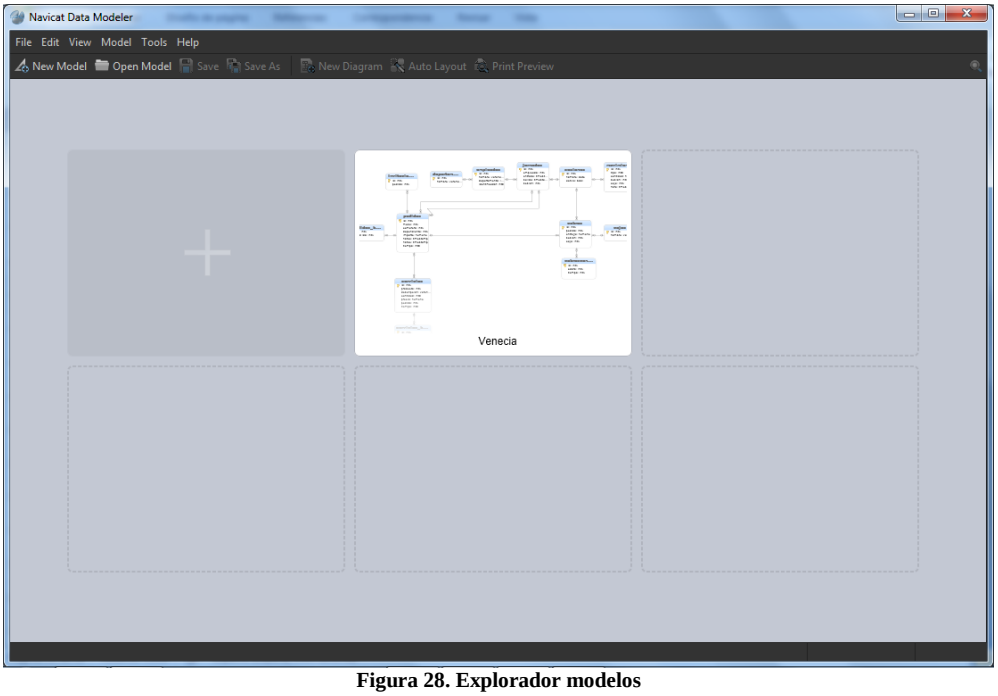


Figura 28. Explorador modelos

Con esta interfaz se pueden crear nuevos modelos o abrir los modelos más recientes. Al hacer "click" en alguno de los modelos se abrirá la ventana de edición de modelos que veremos a continuación en la Figura 29.

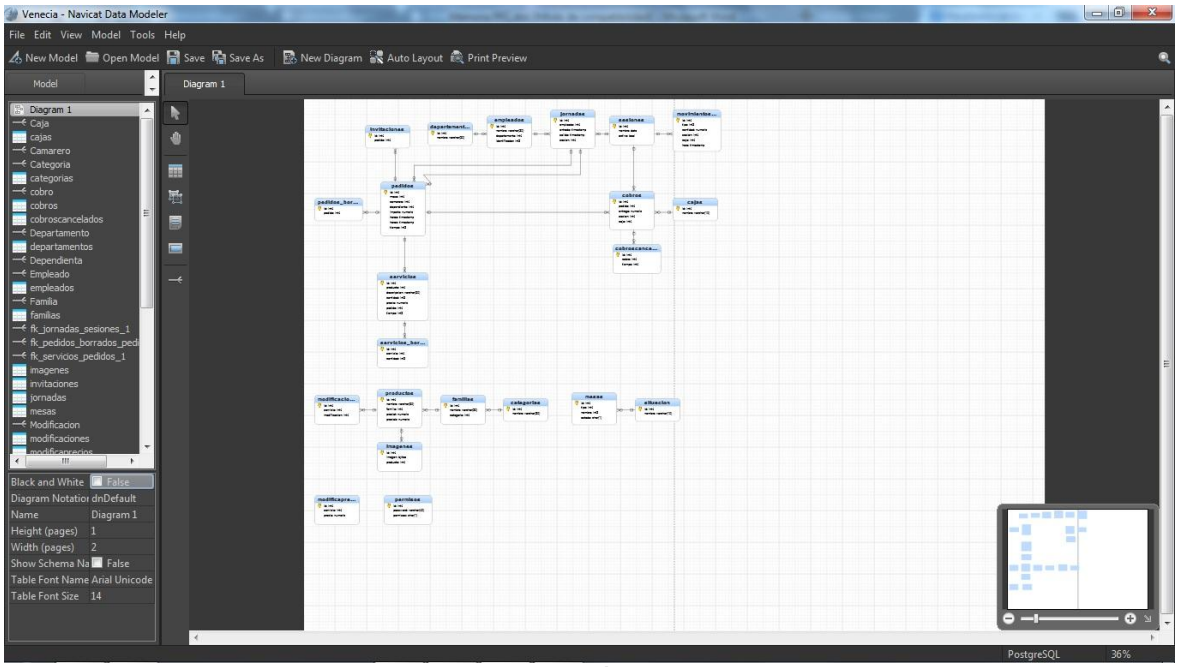


Figura 29: Edición de modelos

En la ventana anterior se observa cómo, de una forma sencilla y bastante intuitiva, la aplicación permite diseñar bases de datos utilizando el modelo relacional. Permite crear entidades con atributos y establecer relaciones entre diferentes entidades.

3.3.2.3. Eclipse

Para el desarrollo de sistemas de software es habitual el empleo de entornos de desarrollo integrado, más conocidos como IDEs de las siglas en inglés.
Los IDEs hacen más sencilla la tarea de escribir y corregir el código, principalmente a nivel de sintaxis.

El entorno de desarrollo integrado escogido es Eclipse. Un entorno también explorado superficialmente a lo largo de la carrera.

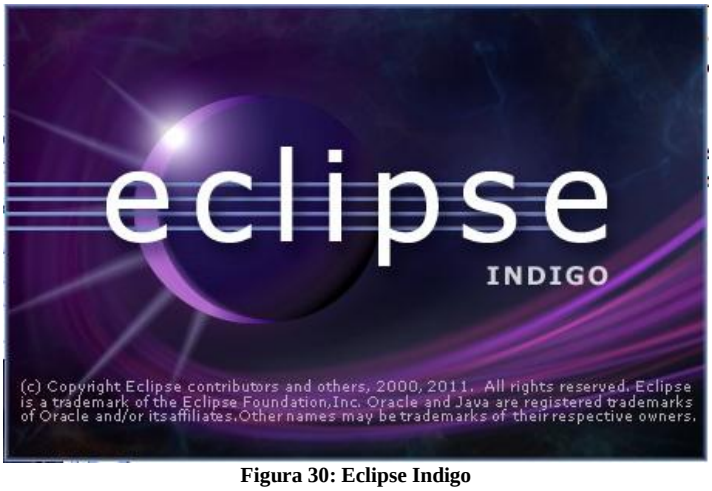


Figura 30: Eclipse Indigo

A pesar de que eclipse cuenta con numerosos plugins que dotan a este IDE de nuevas funcionalidades, para este proyecto no se ha empleado ningún plugin específico. Aunque,

como se puede ver en la figura 31, se encuentra instalado el plugin que facilita la programación para la plataforma Android, ya que se empleará para crear la aplicación telemática que emplearán los trabajadores para gestionar remotamente los pedidos de los clientes. Pero eso no forma parte del alcance de esta fase del proyecto general (como se puede ver en la motivación y objetivos).

El entorno de trabajo utilizado es el siguiente:

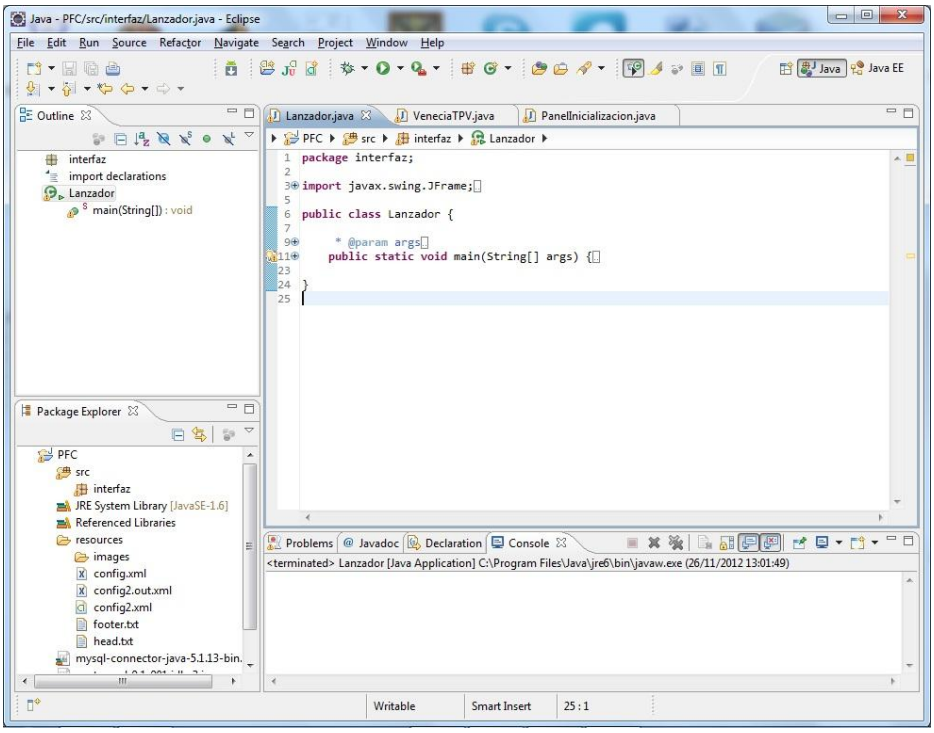


Figura 31: WorkSpace Eclipse

Como se puede observar, el espacio de trabajo que este IDE presenta:

- Barra superior de herramientas:
  Con una barra de menú que permite guardar proyecto, compilar, ejecutar, etc… Y una barra que contiene accesos directos a estas opciones.

- Explorador de clase:
  En este marco se visualizan las variables y métodos de la clase que se encuentre seleccionada para editar.

- Explorador de proyecto:
  En este marco se visualiza el árbol de archivos del proyecto, desde el cual, con un doble click, se puede abrir cualquier archivo del proyecto.

- Editor de texto:
  En la parte central se puede ver la ventana principal del IDE, esta ventana permite visualizar y editar el texto de cada clase u archivo en uso. Este editor de texto ofrece ciertas herramientas que facilitan las tareas del programador, como pueden ser: opciones de autocompletado, autocorrección, señalización de las palabras reservadas, señalización de errores de sintaxis, etc.

- Panel de depuración y consola:
  Por último, en la parte inferior se puede ver el panel de depuración y consola, en dicho panel se mostrará información referente al comportamiento de la aplicación en tiempo de ejecución, permitiendo con ello encontrar los errores que pueden existir en el código o realizando comprobaciones sobre el buen funcionamiento del mismo.

Esta es la distribución del espacio de trabajo utilizada para el desarrollo de este proyecto, sin embargo, cabe destacar que el espacio de trabajo es totalmente configurable en función de las preferencias de cada programador.

3.3.2.4. JVM

Como se ha comentado anteriormente, la implementación de la aplicación se realizará utilizando el lenguaje de programación Java. Esta determinación hace ineludible el uso de una máquina virtual Java.
Sin el uso de dicha máquina, los programas implementados en Java no pueden funcionar.
A la máquina virtual de Java se la denomina Java Runtime Environment (JRE).
Para este proyecto se ha empleado la versión 6 del JRE.

Al crear cualquier proyecto en Eclipse hay que especificar el JRE a usar para dicho proyecto. El JRE hay que añadirlo a Eclipse a través del menú de preferencias:

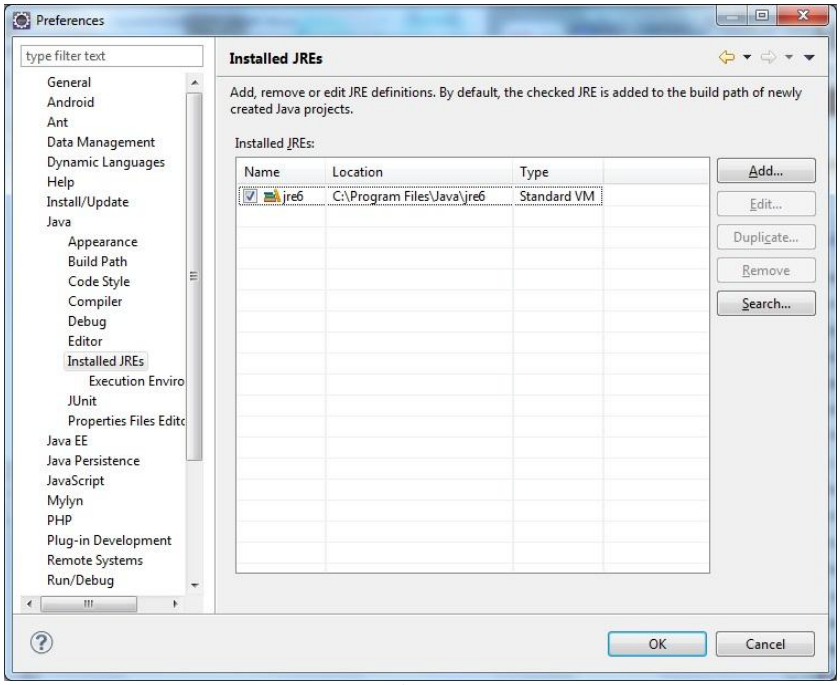


Figura 32: Preferencias de Eclipse

De esta forma, cada vez que se crea un proyecto y se ejecuta, Eclipse emplea el JRE indicado.

3.3.3. Librerías utilizadas.

A continuación se describirán las librerías externas que se han utilizado.

3.3.3.1. Java.sql y JDBC

Java.sql provee de una API para acceder y procesar los datos almacenados en una fuente utilizando el lenguaje de programación Java. Dicha fuente normalmente es una base de datos relacional. Esta API incluye un marco de trabajo donde se pueden instalar diferentes drivers para acceder a diferentes fuentes de datos.
Aunque la API JDBC se utiliza fundamentalmente para enviar sentencias SQL a las bases de datos, permite leer y escribir datos en cualquier fuente de datos con formato tabular.

De todas las clases que esta API contiene, únicamente se incorporarán las cuatro siguientes:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
```

Sin embargo, esta API no es suficiente para poder acceder a la base de datos. Es necesario también instalar el driver correspondiente, como se ha comentado anteriormente.
Para el caso que nos ocupa, PostgreSQL como sistema gestor de base de datos, se ha empleado el driver "*postgresql-9.1-901.jdbc3*". Este driver, añadido al proyecto, permite acceder desde Java a este tipo de sistemas gestores de bases de datos.

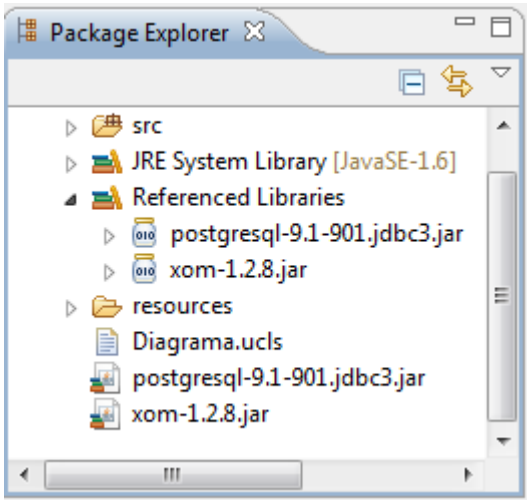


Figura 33: Package Explorer

3.3.3.2. XOM

Como se puede ver en la figura 33, también se emplea la librería XOM. Esta librería ofrece una API para Java que da soporte al tratamiento de XML: parseo, búsquedas, modificación, generación y serialización.

```
import nu.xom.*;
```

Se ha escogido XOM en lugar de alguna de las otras opciones disponibles como JDOM, SAX o DOM, debido a sus principios básicos: corrección, simplicidad y rendimiento.
Corrección en cuanto al modelado XML. XOM modela XML completamente y correctamente, esto ha sido lo más importante que se ha tenido en cuenta para su diseño. Otros APIs comprometen parte del modelado para lograr un mejor rendimiento.
Simplicidad porque cada método hace exactamente lo que se supone que debe hacer en función de su nombre y el número de métodos públicos son los estrictamente necesarios para cumplir con la funcionalidad requerida. Esto hace que sea fácil de aprender y usar.
En cuanto al rendimiento, XOM se ha diseñado para ser rápido para los usos más habituales y a su vez los documentos XOM son hasta 3 veces menores en memoria que los documentos creados empleando otras APIs.

3.3.3.3. Java.security

Para implementar el módulo de seguridad de la aplicación, es decir, mantener ciertas partes restringidas a usuarios privilegiados se hace uso de la librería *Java.security*. Más concretamente, dentro de esta, se utiliza:

```
import java.security.MessageDigest;
```

La clase *MessageDigest* dota a la aplicación de un algoritmo de generación de resúmenes de mensajes, tales como MD5 o SHA. Los resúmenes de mensajes son hash calculados de forma unidireccional, esto es, la operación no se puede calcular a la inversa. Del hash no se puede obtener el mensaje original. Esto hace que sean altamente utilizados para almacenar contraseñas en bases de datos, ya que aún recuperando el hash no se podría obtener la contraseña.
En el caso de esta aplicación la contraseña de privilegios se encuentra almacenada en la tabla "*permisos*" de la base de datos. El campo "*password*" de esta tabla almacena el hash de la contraseña original, calculado mediante el algoritmo SHA-1. Se genera un mensaje de longitud 40, independientemente de la longitud de la contraseña original.

3.3.3.4. Otras librerías

Como no podía ser de otra manera, también se han utilizado otras librerías que Java incluye por defecto y son de uso común en la mayoría de proyectos Java.
Aunque *Java.sql* y *Java.security* también son librerías que vienen por defecto en Java, al no ser tan comunes y ser de propósito específico, y además tener una importancia reseñable en este proyecto, se ha decidido dedicarles un apartado a cada una.
Las librerías comunes de Java que se han empleado son:

- *Java.io*. Librería básica para operaciones de entrada y salida de datos. Ya sea salida por pantalla o a archivo, o entrada por teclado, archivo, etc…
- *Java.net*. Librería que implementa las funciones básicas para realizar aplicaciones que hagan uso de la red.
- *Java.util*. Librería que ofrece ciertas funciones misceláneas que pueden resultar de utilidad. En nuestro caso se ha utilizado la clase Calendar para obtener para obtener la fecha.
- *Java.awt*. Contiene todas las clases para crear interfaces de usuario y para pintar gráficos o imágenes.

- *Java.text.DecimalFormat*. Clase empleada para representar los números en forma decimal y redondearlos para obtener cifras válidas monetariamente hablando.
- *Javax.imageio.ImageIO*. Es una clase que contiene los métodos *ImageReader* e *ImageWriter* y realiza codificaciones y decodificaciones simples.
- *Javax.swing*. Proporciona un conjunto de componentes ligeros que, en el máximo grado posible, son válidos para todas las plataformas.

3.3.4. Decisiones tomadas

En este apartado se señalarán especialmente las decisiones tomadas sobre el diseño o implementación de la aplicación.

3.3.4.1 Base de datos

El Sistema de Gestión de Bases de Datos escogido para la implementación de la base de datos necesaria para este proyecto es PostgreSQL.
Los motivos por los que se ha optado por esta alternativa son los siguientes:

- Se trata de un sistema de código abierto, y por lo tanto gratuito, frente a otras opciones de pago como Oracle, Fox Pro o Microsoft Access.
- Es un sistema ampliamente extendido, con buenas referencias y en constante desarrollo.
- Es un sistema que soporta una alta concurrencia y permite que mientras un usuario escribe en una tabla, otros accedan sin necesidad de bloqueos.
- Trabaja a una velocidad constante, independientemente de la cantidad de información que contenga la base de datos.
- Utiliza compresión para almacenar los datos, lo que reduce la necesidad de espacio en disco y las lecturas sobre archivos son mucho más rápidas.
- Permite indexación parcial.
- Soporta una completa API asincrónica para el uso de las aplicaciones cliente. Lo que puede aumentar el rendimiento en algunos casos hasta un 40%.
- Cumple los principios ACID (Atomicidad, Coherencia, Aislamiento y Durabilidad) lo que garantiza la integridad de los datos.
- Permite la comprobación de limitaciones al insertar o modificar datos. Esto permite que todos los datos introducidos sean acordes a las reglas de diseño de las tablas.
- Soporta *triggers* en función de los cambios en las tablas. Los *triggers* son disparadores que permiten ejecutar cualquier función definida por el usuario al cumplirse alguna condición específica.
- Es modular en su diseño, y aunque la replicación no va incorporada en su núcleo, existen módulos que otorgan esta funcionalidad. Como se detallará a continuación, en este proyecto, se empleará Rubyrep.
- Soporta un amplio abanico de tipos de datos.
- Permite la realización de subconsultas.

La versión de PostgreSQL empleada durante este proyecto es la versión 9.1.3.

3.3.4.2 Replicación de la base de datos.

Debido a que PostgreSQL no ofrece replicación entre bases de datos, se hace necesario recurrir a alguna herramienta externa para ofrecer este servicio. En este caso se ha recurrrido a Rubyrep.

Rubyrep es una herramienta de replicación de bases de datos del tipo master-master. Se ha escogido esta herramienta, fundamentalmente, por su sencillez.
Sus características principales son:

- Facilidad de uso. Se ha definido como "ridículamente fácil de usar".
- Soporta PostgreSQL y MySQL.
- Opera entre dos bases de datos, refiriéndolas como "left" y "right".
- Configuración muy simple, mediante un solo archivo de configuración se establecen todos los parámetros necesarios.
- Únicamente hay que descargar y extraer los archivos para poder comenzar a funcionar.
- Compatible con Unix y Windows.
- El diseño de la base de datos y las tablas es independiente, no hay que indicarle nada a Rubyrep.
- Proporciona tres servicios: compara, sincroniza y replica los datos.
- Necesita una Máquina Virtual de Java (JVM) instalada en el equipo.

La versión empleada de Rubyrep es la 1.2.0.

Para el proyecto que nos ocupa, con una replicación entre dos bases de datos es suficiente. Sin embargo, Rubyrep permite realizar replicaciones empleando un esquema en forma de estrella, lo que permite ampliar el número de bases de datos replicadas sin problema. Se haría de la siguiente manera:

Si se quiere replicar las bases de datos A, B y C se operaría de la siguiente manera:

- A se replica con C.
- B se replica con C.

De esta manera podría incrementarse el número de nodos replicados.

3.3.4.3 Lenguaje de programación

El lenguaje de programación escogido para el desarrollo del proyecto es JAVA.
Se ha escogido dicho lenguaje por sus características:

- Es un lenguaje de alto nivel.
- Es un lenguaje orientado a objetos
- Las aplicaciones creadas en Java corren sobre una máquina virtual.
- Las aplicaciones creadas corren en cualquier sistema operativo con una máquina virtual compatible instalada.

También se ha escogido este lenguaje de programación con fines didácticos. A lo largo de la carrera se ha estudiado el lenguaje Java así como el paradigma de la programación

orientada a objetos. Sin embargo, con este proyecto se pretende avanzar y profundizar en el conocimiento de este paradigma de la programación, a la vez que se profundiza en el conocimiento del lenguaje Java en particular.

3.3.4.4 Recursos

Se han definido una serie de recursos para la aplicación tales como las imágenes utilizadas en los botones, un archivo de configuración y dos archivos auxiliares para la creación de los tiquets.

Las imágenes que se muestran en los botones se encuentran organizadas dentro de la estructura de directorios de recursos de la aplicación. Dentro del directorio de recursos, se encuentra el directorio "*images*", y dentro del mismo se hallan otros dos directorios "*interfaz*" y "*productos*". En el directorio "*interfaz*" se encuentran las imágenes correspondientes al diseño de los botones propios de la interfaz, mientras que en el directorio "*productos*" se encuentran las imágenes de los productos.
Si se añade un nuevo producto a la base de datos y no se añade ninguna imagen correspondiente en el directorio de productos, no se mostrará ninguna imagen, en su lugar, solo se mostrará el nombre del producto. En caso de añadir en el directorio de productos una imagen en formato "*.jpg*" con el nombre del producto, esta se mostrará en la casilla correspondiente.

Para dotar de persistencia a las opciones de configuración se ha optado por utilizar un archivo en formato "*.xml*" donde se almacenarán. La aplicación, al arrancar, obtendrá los valores de configuración de dicho archivo y la ejecución de la aplicación se realizará en consecuencia, teniendo en cuenta los datos previamente almacenados.

Dentro de las opciones de configuración existen las opciones para modificar la cabecera y el pié de los tiquets. Para esto, se utilizan los archivos auxiliares de creación de tiquets. Estos consisten en dos archivos de texto que contienen el texto a mostrar en cada sección. Se ha optado por esta manera porque ofrece una solución que permite modificar dichas secciones desde el exterior de la aplicación si se considera oportuno y además se podrían migrar entre distintos equipos sin problemas.

3.3.4.5 Redimensionado de imágenes

Se ha decidido que para facilitar la tarea de los usuarios al insertar las imágenes la aplicación redimensiona automáticamente las imágenes. De esta forma, con simplemente insertar la imagen en el directorio correcto la aplicación se encargará de otorgarle el tamaño correcto.

Para esto, se emplea la clase icono, la cual consta del método "*getScaledInstanced*". Este método devuelve una versión redimensionada de la imagen que se le introduce como argumento en forma de "*BufferedImage*". Este método ofrece varias formas de realizar la reducción de la imagen, correspondiéndose con diferentes niveles de calidad del resultado en función de las distintas situaciones iniciales (tamaño de imagen original mayor o menor del tamaño objetivo) de las imágenes que se pueden encontrar.

Como parámetros de entrada, este método recibe: un "*BufferedImage*" correspondiente a la imagen a redimensionar; dos "*int*" indicando el ancho y el alto a los cuales se quiere

obtener la imagen; un "*object*" que indica el tipo de renderizado (puede ser: interpolación, interpolación al más cercano, interpolación bilineal e interpolación bicúbica); y, por último, un "*boolean*" para indicar si se quiere obtener un resultado de mayor calidad, lo que conllevaría mayor tiempo de cómputo.

3.3.4.6 Seguridad

Como método de seguridad para restringir las opciones pertinentes se ha empleado un acceso mediante contraseña. Las contraseñas se hayan almacenadas en la base de datos. Tal y como se ha explicado en la explicación del uso de la librería "*Java.security*".

El sistema está diseñado de forma que a cada contraseña se le indica su nivel de privilegios, de esta forma pueden existir diferentes contraseñas que den acceso a distintas funcionalidades.
Para el acceso a las funciones se solicita la introducción de una contraseña. En caso de que la contraseña introducida sea válida, se dará acceso a la sección solicitada. Si la contraseña no es válida, o si la contraseña no tiene suficientes permisos para acceder, el acceso será denegado.

En cuanto a la seguridad de la red, se empleará una red con acceso protegido mediante clave WPA2, lo que proporcionará seguridad en cuanto a acceso a la red y, a su vez, proporcionará seguridad ante la utilización de "*sniffers*" por parte de terceros mediante encriptación.
Sobre el posible acceso a la red a través de internet, aún no está definido si la red tendrá alguna puerta de enlace con acceso a internet. Por lo tanto, en el momento de instalación final se actuará en consecuencia.

3.3.4.7 Eventos

Esta aplicación es una aplicación puramente visual, por lo tanto, los eventos generados por los usuarios son la fuente principal de "movimiento" en la aplicación. Para el trato de los eventos se emplea la interfaz "*ActionListener*".
Esta interfaz permite, mediante la implementación de la clase "*ActionPerformed*", registrar los eventos que suceden y actuar en consecuencia. Para ello, los componentes que puedan generar eventos deben registrarse mediante el método "*AddActionListener*". Cuando se produce un evento, mediante el método "*getSourc*e()" del evento registrado, se identifica el componente que generó el evento, y se ejecutan las acciones predefinidas para dicho evento.

Se han empleado varios "*listeners*". Aunque al principio se comenzó utilizando un único "*listener*" incluso para los eventos ocurridos en ventanas emergentes, con el avance del desarrollo se decidió crear un "*listener*" por cada nueva ventana utilizada, con el fin de no sobrecargar demasiado el principal. Sin embargo, no se modificaron los eventos pertenecientes a las ventanas anteriormente implementadas. Por lo tanto, aún quedan ventanas sin "*listener*" propio. Esto, será corregido en futuras actualizaciones de la aplicación por cuestión de tiempo.

3.3.5. Codificación.

En este apartado se destacará una pequeña muestra del código empleado para implementar cierta funcionalidad de la aplicación.

3.3.5.1 Conexión con la base de datos

```
      conexion_db=DriverManager.getConnection("jdbc:postgresql://localho
st:5432/venecia","postgres","postgrespass");


      statement=conexion_db.createStatement(ResultSet.TYPE_SCROLL_INSENS
ITIVE, ResultSet.CONCUR_READ_ONLY);

      statement2=conexion_db.createStatement(ResultSet.TYPE_SCROLL_INSEN
SITIVE, ResultSet.CONCUR_READ_ONLY);

      statement3=conexion_db.createStatement(ResultSet.TYPE_SCROLL_INSEN
SITIVE, ResultSet.CONCUR_READ_ONLY);
```

Con este código se establece la conexión a la base de datos empleando en este caso el conector jdbc para postgresql. Una vez establecida la conexión se generan tres "statement" que se utilizarán como puente y permitirán realizar consultas a la base de datos.

3.3.5.2 Ejecución de consultas SQL

Una vez se ha definido la conexión con la base de datos y se han establecido los "canales" se puede comenzar a realizar consultas, ediciones o inserciones en la base de datos:

```
if(a==0){
      query="Update pedidos set camarero="+aux.get_index()+" where
id="+current_pedido+" and horac is null";
}else{
      query="update pedidos set dependienta="+aux.get_index()+" where
id="+current_pedido+" and horac is null";
}
try{
      statement2.executeUpdate(query);
}catch (Exception e){
      e.printStackTrace();
}
```

3.3.5.3 Ejecución de Rubyrep

Rubyrep es una aplicación independiente de este proyecto, que simplemente es utilizado. Para ello, se ejecutará el comando de Windows correspondiente desde Java.

```
if(obj_conf.bol_replicante){
      try
      {
      String[] comando=new String[]{"cmd","/c","C:/rubyrep-
1.2.0/rubyrep","replicate","-c","C:/rubyrep-1.2.0/Replica.conf"};
```

```
      p = Runtime.getRuntime().exec(comando);
      }
      catch (Exception e)
      {
      JOptionPane.showMessageDialog(this,"Error al intentar iniciar la
replicación");
      }
}
```

Mediante este código se comprueba que la configuración especifica que el equipo local es el replicante, en caso de serlo se crea el comando de iniciación de la replicación y se ejecuta empleando la clase *Runtime*, la cual crea un nuevo proceso.

3.3.5.4 Manejo de eventos

Como se ha comentado en el apartado de eventos anteriormente, para poder responder a los posibles eventos producidos por pulsaciones de botones, es necesario registrar los componentes en un "*ActionListener"*.
Para ello, se debe implementar la interfaz "*ActionListener"*, tal y como se ve a continuación:

```
public class VeneciaTPV extends JFrame implements ActionListener
```

Sobreescribiendo el método "ActionPerformed" se implementan las acciones a realizar en función del objeto que produce el evento.

```
public void actionPerformed(ActionEvent ev) {
      if (ev.getSource()==Bt_barra){
            ((CardLayout)Pa_terbar.getLayout()).show(Pa_terbar,"barra");
            encontrado=true;
      }
}
```

En el ejemplo anterior se muestra cómo gestionar la pulsación del botón *Bt_barra*. Sin embargo, para que se active el método tras pulsar el botón, previamente debe estar registrado o añadido el botón al "*listener*".
Para ello, tras construir todos los componentes se añaden adecuadamente al "*listener"* que se encarga de gestionar sus eventos.

```
Bt_barra=new JImageButton("<html><p align='center'>Barra</p></html>");
Bt_barra.setFont(ft_bot_general);
Bt_barra.setBackground(marron);
Bt_barra.setForeground(Color.white);
Bt_barra.setBorderPainted(pintar_bordes);
Bt_barra.addActionListener(this);
```

3.3.5.5 Creación de archivo xml

Para gestionar el archivo de configuración, el cual es de extensión xml, es necesario crear una estructura de elementos según el formato xml. Esto se ha realizado de la siguiente manera:

```
        Element raiz = new Element("config");
```

```
        Element caja= new Element("caja");
        if(combo_caja.getSelectedIndex()==0){
            caja.appendChild("1");
        }else{
            caja.appendChild("2");
        }
        raiz.appendChild(caja);

        Element replicante= new Element("replicante");
        if(cb_replica.isSelected()){
            replicante.appendChild("y");
        }else{
            replicante.appendChild("n");
        }
        raiz.appendChild(replicante);

        Element modif= new Element("modif");
        if(cb_modif.isSelected()){
            modif.appendChild("y");
        }else{
            modif.appendChild("n");
        }
        raiz.appendChild(modif);

        Element imp_tiq= new Element("imp_tiquet");
        imp_tiq.appendChild(tf_imp_tiq.getText());
        raiz.appendChild(imp_tiq);

        Element imp_com= new Element("imp_comanda");
        imp_com.appendChild(tf_imp_com.getText());
        raiz.appendChild(imp_com);

        Element cabecera=new Element("cabecera");

            cabecera.appendChild(ta_head1.getText());
        raiz.appendChild(cabecera);
```

Una vez se ha creado un elemento raíz del que cuelgan el resto de elementos del documento se procede a crear el documento.

```
Document doc = new Document(raiz);
try{
      Serializer serializer = new Serializer(System.out, "UTF-8");
      serializer.setIndent(4);
      serializer.setMaxLength(64);

      serializer.setOutputStream(new BufferedOutputStream(new
FileOutputStream("resources/config.xml")));
      serializer.write(doc);
      serializer.flush();
}catch (Exception e){
      e.printStackTrace();
}
```

4. Instalación y configuración

Al tratarse de una parte de la aplicación, la instalación de esta parte no dará por concluido el proceso completo final, sin embargo, sí que se pueden instalar algunas partes de manera definitiva.

Se ha dispuesto un archivo de extensión ZIP que contiene todos los archivos y la estructura de directorios, necesaria para la correcta localización de los recursos utilizados por la aplicación, y para realizar la instalación de la forma más sencilla posible.

Tras descomprimir el archivo en la ruta seleccionada para la instalación y realizar los sencillos pasos que se describen a continuación, la aplicación estará lista para ser utilizada.

4.1. Instalación del SGBD e instanciación de la base de datos.

Uno de los procesos que se deben llevar a cabo es la instalación de la base de datos. Como se ha comentado anteriormente, en este caso se ha decidido utilizar un SGBD PostgreSQL en su versión 9.1.
A partir de la versión 8.4 se puso a disposición de los usuarios un instalador visual sencillo, con la finalidad de facilitar el proceso de instalación. Por lo tanto, para realizar la instalación basta con seguir los pasos que se van indicando y no revierte mayor problema.

Una vez realizada la instalación se debe crear la propia base de datos a utilizar en la aplicación, así como otorgarle la estructura necesaria. Para esto se ha creado el archivo InstallDB.bat.
Este script se encarga de ejecutar los comandos necesarios para ejecutar, sobre el SGBD y después sobre la base de datos, los scripts InstallDB.sql y FillDB.sql, respectivamente.
El script InstallDB.sql recoge los comandos SQL necesarios para la creación de la base de datos. Y el script FillDB.sql recoge los comandos SQL necesarios para crear las tablas y secuencias necesarias, además de añadir el contenido mínimo básico que contendrán.

4.2. Configuración de RubyRep

Tras tener preparada la base de datos se preparará el complemento para realizar la replicación de la misma. Como anteriormente se ha comentado dicho complemento se corresponde en este caso con Rubyrep en su versión 1.2.
Rubyrep consta de dos versiones, una versión estándar, basada en Ruby, y otra versión conocida como JRuby, la cual utiliza el motor JAVA. Debido al mejor rendimiento en cuanto a velocidad y a la menor complejidad de instalación, se ha utilizado la versión JRuby.
Para instalar dicha versión, basta con descargar un ZIP desde su página web oficial y descomprimirlo en el directorio donde se quiera realizar la instalación. Sin embargo, dicho paquete descomprimido se halla añadido al paquete de instalación, por lo tanto, únicamente será necesario realizar la configuración del mismo y copiar el directorio en la ruta C:\ del equipo
.
La estructura del dicho archivo de configuración es la siguiente:

```
RR::Initializer::run do |config|
 config.left = {
  :adapter  => 'postgresql', # or 'mysql'
  :database => 'databasename',
  :username => 'databaseusername',
  :password => 'databasepassword',
  :host     => 'databasehostIP'
 }

 config.right = {
  :adapter  => 'postgresql',
  :database => 'databasename',
  :username => 'databaseusername',
  :password => 'databasepassword',
  :host     => 'databasehostIP'
 }

 config.options[:sync_conflict_handling] = :right_wins
 config.add_table_options 'emp', :sync_conflict_handling => :left_wins

 config.include_tables /./
end
```

Una vez los datos de configuración se han establecido es posible iniciar la replicación de la base de datos aplicando el siguiente comando en la ruta de Rubyrep.

```
Rubyrep replicate –c archivo_configuracion.conf
```

Aunque en este caso, es la aplicación la que se encarga de iniciar la replicación, por lo tanto, no será necesario utilizar el comando manualmente.

5. Líneas de desarrollo futuras

Debido a la falta del hardware, no se ha podido testear el funcionamiento de la impresora de tiquets. Por esto, la completa implementación de esta parte queda pendiente para realizar durante el proceso de instalación en el escenario real.
Las particularidades de los diferentes comandos que utilizan las distintas impresoras para realizar tareas tales como cortar el papel, abrir el cajón de dinero, o cambiar la fuente de la letra, hacen que sea necesario disponer del hardware para realizar la adaptación del código a los comandos correctos.

Como se ha comentado en el capítulo de introducción, la aplicación desarrollada durante este proyecto se complementa con los dos siguientes programas a realizar: el programa de gestión de datos y la aplicación móvil para equipos portátiles. Por lo tanto, como primera tarea dentro de las líneas de desarrollo futuras se encuentra el desarrollo e implementación de dichas aplicaciones.

A su vez, para la correcta incorporación de ambas aplicaciones puede ser necesario realizar una adaptación de la aplicación realizada. Implementando nuevas funcionalidades o cambios estructurales, en función de las nuevas necesidades surgidas tras la incorporación de dichas aplicaciones.

Una vez se hayan desarrollado las demás aplicaciones, se realizará una instalación y configuración completa de la red, adecuándola tanto a las necesidades que se requieran como al ambiente del lugar (saturación Wi-Fi, exposición a internet, etc).

Además, como toda aplicación en fase de explotación, se irán implementando actualizaciones con mejoras cuando sea necesario, además de revisiones de algoritmos en busca de una mejora de la eficiencia de la aplicación.
Por supuesto, para llevar a cabo las mejoras o las nuevas adaptaciones a las necesidades será necesario mantener el contacto y dar soporte a la empresa cliente de la aplicación.

Posteriormente, tras el completo desarrollo, y la comprobación en un escenario real, de la correcta funcionalidad de la aplicación se pretende distribuir la aplicación entre nuevos clientes. Realizando las adaptaciones necesarias en función de las necesidades de los nuevos clientes.

6. Bibliografía

[1] Eckel, B. Thinking in Java, 4th edition. Prentice Hall, Engelwood Clifts, 2006.

[2] Schildt, H. Java 7: The complete Reference, 8th edition. McGraw-Hill Osborne, Nueva York, 2010.

[3] Jiménez A., Pérez F. Aprende a programar con Java. Paraninfo, 2012.

[4] Eisentraut P., Helmle B. PostgreSQL-Administration. O'Reilly, 2008

[5] Sitio Web: Obtener eclipse y su documentación. www.eclipse.org

[6] Sitio Web: Obtener PostgreSQL y su documentación www.postgresql.org

[7] Sitio Web: Obtener Rubyrep y su documentación www.rubyrep.org

[8] Sitio Web: Obtener XOM y su documentación www.xom.nu

[9] Sitio Web: Obtener conector JDBC para PostgreSQL http://jdbc.postgresql.org/

[10] Sitio Web: Referencia de Java http://docs.oracle.com/

[11] Sitio Web: Obtener Navicat www.navicat.com

[12] Sitio Web: Plugin de eclipse www.objectaid.com

[13] Sitio Web: www.firesmax.es

[14] Sitio Web: www.softpyme.net

[15] Sitio Web: www.gestiontpv.com

[16] Sitio Web: www.floreantpos.com

[17] Sitio Web: www.posper.net

[18] Sitio Web: www.laun4j.sourceforge.net