UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Uso de modelos de programación asíncronos administrados por eventos y bases de datos NoSQL en sistemas de puntos de venta para PyMES

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Marco Antonio García Téllez

DIRECTOR DE TESIS

Ing. Jorge Alberto Rodríguez Campos

Ciudad Universitaria, Cd. Mx., 2019

Agradecimientos

A mis padres, Juan y María del Carmen, que con su esfuerzo, dedicación y cariño me enseñaron que la vida está llena de retos y que para alcanzar las metas, siempre hay que esforzarse y dar lo mejor de uno mismo, este logro es para ustedes, gracias por todo.

A mi hermano Juan Carlos, con quien compartí los mejores años de mi infancia, en algún momento de la eternidad nos volveremos a encontrar.

A mis hermanos Leticia y Luis, gracias por su compañía y apoyo incondicional, me motivaron a no rendirme y concluir este proyecto.

A mi querida esposa Ana Cristina que me dio una maravillosa familia, gracias por tu apoyo, comprensión y compañía he vivido las mejores experiencias, cada momento a tu lado me recuerda que todo el esfuerzo ha valido la pena.

A mis hijos Samantha y Santiago que son el motor de mí existir, mi principal fuente de motivación para seguir adelante, mi esfuerzo hacia ustedes, es orientarlos y motivarlos para que desarrollen al máximo sus capacidades, se sientan orgullosos de sus logros y sean felices.

A mi director de tesis el Ing. Jorge Alberto, gracias por su paciencia, consejos y apoyo que me brindó durante el transcurso de este proyecto.

A mis familiares, que con su apoyo, compañía y amistad han hecho que este camino recorrido sea más ameno, gracias por confiar en mí.

A mis profesores y sinodales gracias por haberme compartido una parte de sus conocimientos y experiencias.

A la Facultad de Ingeniería de la UNAM, gracias por brindarme un lugar para mí formación académica y profesional.

A mis compañeros y amigos por que siempre han sido y serán parte de este viaje.

Contenido

Introducción

El presente trabajo de tesis, tiene como finalidad desarrollar una alternativa a los sistemas comerciales de punto de venta para restaurantes que actualmente existen. Las principales características de estos sistemas son el alto costo, escasa portabilidad, escalabilidad y que además suelen ser complicados de utilizar para personas con conocimientos mínimos en sistemas.

Se utilizará principalmente el entorno de ejecución Node.js, el cual está construido con el motor V8 de JavaScript desarrollado por Google. MongoDB es el sistema de base de datos NoSQL orientado a documentos que se utilizará para almacenar la información.

Todo esto se integra con archivos en formato JSON para transportar la información entre las distintas capas de la aplicación y con el ya mencionado lenguaje de programación JavaScript. Estas herramientas ofrecen una solución completa a la problemática planteada. Cabe mencionar que este sistema está enfocado a micro, pequeñas y medianas empresas.

En el **capítulo 1**, se indaga sobre los antecedentes y la situación actual de los sistemas de puntos de venta. Se desarrolla el análisis de requerimientos que son necesarios para implementar la solución adecuada ante dicha problemática.

Para dar solución a la problemática planteada, en el **capítulo 2,** se documentan las herramientas con las que se trabajará en esta tesis, así como los conceptos y definiciones de los paradigmas de programación utilizados en los frameworks asíncronos. Se describe el concepto de bases de datos NoSQL y se explica las ventajas de este tipo de tecnologías para el entorno emergente del *Big Data*.

En el **capítulo 3**, se establecen las bases técnicas para el desarrollo de la solución. Se define el alcance del proyecto, dando respuesta a los objetivos propuestos al inicio de la documentación. Otro aspecto a cubrir en este capítulo es la descripción a detalle de la arquitectura del sistema, diversificando varios escenarios en los que se puede funcionar el sistema, como por ejemplo: la topología en red de 1 ó N impresoras de tickets.

Para **capítulo 4**, se describe la implementación y la funcionalidad de la aplicación ya en un ambiente productivo, se explica cada módulo del sistema y las reglas de negocio.

Se detalla el funcionamiento por capas de la aplicación y el funcionamiento e implementación del modelo asíncrono en Node.js.

En el **capítulo 5**, una vez documentada la parte teórica y haber implementado el sistema, se realiza un análisis a fondo sobre los resultados obtenidos en producción, se realizan pruebas unitarias y funcionales, demostrando la eficiencia de los frameworks mencionados, verificando que cada módulo funcione adecuadamente de acuerdo al alcance y comprobar la respuesta en tiempo real de las solicitudes hechas por los usuarios.

En el **capítulo 6**, se propone un desarrollo a futuro de aplicaciones nativas para dispositivos móviles que interactúen directamente con el framework Node.js, sin necesidad de realizar cambios en el lado del servidor, con ello se puede realizar una pequeña comparativa de cómo funciona el sistema en aplicaciones nativas versus explorador web.

También se mencionan otras características, como el mejoramiento del módulo de inventarios, recetarios, análisis y minería de datos.

En el **capítulo 7**, se presentan las conclusiones obtenidas durante el desarrollo de este proyecto de tesis.

Finalmente en el **capítulo 8** se incluye bibliografía y mesografía sobre los cuales se fundamentó esta investigación.

Capítulo 1

Planteamiento del problema y análisis de requerimientos

1. Planteamiento del problema y análisis de requerimientos

1.1 ¿Qué es un sistema de punto de venta?

Los sistemas de puntos de venta, son una parte fundamental en cualquier negocio que se dedique a la comercialización de productos, como, por ejemplo: comida, ropa, zapatos, abarrotes, aparatos electrónicos, etc.

Para el caso de los negocios que se dedican a la venta de comida, si el restaurante tiene una gran afluencia de clientes, es imposible no contar con un sistema que sea capaz de registrar y soportar todas las operaciones que ocurren durante el día. El sistema tiene que ser sencillo y fácil de usar por el personal, tiene que ser rápido y eficiente para responder a las solicitudes en tiempo real y sobre todo tiene que ser escalable para agregar nuevos módulos que se requieran implementar en el futuro.

Actualmente es indispensable el uso de las tecnologías de la información para poder controlar y procesar toda esta cantidad de información generada por las PyMES en sus operaciones diarias: para conocer en tiempo real las ventas totales, tener una mejor gestión sobre el negocio y saber con exactitud los ingresos y egresos.

"Los puntos de venta vienen a automatizar el proceso de salida y cobro de la mercancía en las tiendas departamentales, comercios, restaurantes y otras instituciones. La implementación de los sistemas de punto de venta no son un lujo, sino una necesidad primordial para agilizar los procesos en los que está relacionado la salida de la mercancía en estos tipos de establecimientos."[1].

1.1.1 Panorama en México sobre los puntos de venta

De acuerdo al estudio publicado por el INEGI (Instituto Nacional de Estadística y Geografía) "*Micro, pequeña, mediana y gran empresa, Estratificación de los establecimientos*" del año 2015, cerca del 30 % de toda la actividad de económica, está representada por hoteles y restaurantes, donde el 95% de esta actividad se lleva a cabo por micro empresas y el resto se distribuye entre pequeñas, medianas y grandes empresas.

Esto en unidades económicas significa que hasta ese año en que se publicó el estudio existían aproximadamente 501,448 establecimientos, de los cuales 480,178

[1] www.mbcestore.com.mx

pertenecen a la sección de micro empresas. Para la fecha actual, esta cifra puede ser mucho mayor.

Sin embargo, cerca del 65% de estas empresas cierra operaciones antes de cumplir los 5 años de vida productiva, las razones son diversas, entre las más populares se encuentran, falta de un plan de estrategia comercial, problemas financieros, administrativos y fiscales.

Al ser empresas nuevas y en crecimiento, uno de los factores que deben cuidar es la inversión. En este trabajo se propone crear un sistema de bajo costo, flexible y liviano que se pueda ejecutar en cualquier sistema de bajos recursos, para ayudar a mejorar la administración de las micro, pequeñas y medianas empresas.

En temas posteriores se explicará a detalle los requerimientos y se establecen algunas soluciones a esta problemática.

1.2 Principales funciones y características de los sistemas de puntos de venta para restaurantes

En esta sección se explican las características fundamentales que debe tener cualquier sistema de punto de venta para administrar restaurantes.

- ***Información precisa sobre los pedidos.***

  Es necesario saber qué mesero generó el pedido, la hora exacta en que fue levantado. Éste es un punto fundamental ya que, para mejorar la experiencia del cliente, el pedido no puede demorar demasiado tiempo en prepararse y servirse en la mesa del cliente.

  Saber a qué cliente va dirigido el pedido, tener alguna forma de identificar al cliente como lo son: el número de mesa, cantidad de clientes, vestuario de los clientes, etc.

  Adicionalmente en caso de ser requerido agregar observaciones sobre algún platillo, por ejemplo, el tipo de cocción de la carne: ¾ de cocción, bebidas: con hielo, a temperatura ambiente, etc.

- ***Gestión y orden de los pedidos***

Una vez capturados los pedidos, el sistema de punto de venta gestiona la distribución de los tickets de los pedidos a cada una de las diferentes áreas de trabajo, por ejemplo, si el pedido debe surtirse en la barra de bebidas, el sistema debe agrupar todos los pedidos correspondientes a esta sección e imprimirlos en la impresora de tickets asignada a esta área de trabajo. Para estos casos es necesario disponer de varias impresoras de tickets y situarlas en cada área de trabajo en donde se requiera de una impresora. En el caso de que se disponga de una sola impresora, el sistema debe de imprimir todos los tickets en esta impresora.

Cada pedido debe tener un folio, un consecutivo con el cual las personas encargadas de preparar los pedidos puedan identificar fácilmente el orden en que debe surtirse cada pedido.

- ***Distribución de la información a diferentes dispositivos***

Aunque la información se centraliza en un solo servidor, éste debe de estar disponible y poderse consultar desde cualquier dispositivo que se encuentre dentro de la red local del negocio o inclusive remotamente desde cualquier parte del mundo por medio de internet y en tiempo real.

- ***Manejar diferentes perfiles de usuarios para acceder a la aplicación***

Como cualquier sistema complejo, se rige bajo roles de usuarios, no todos los usuarios tienen las mismas funciones dentro de la aplicación y por seguridad no todos los usuarios pueden visualizar toda la información disponible en el sistema.

Por ello es fundamental que el sistema gestione los diferentes perfiles de acuerdo a las funciones de cada usuario, por ejemplo: en el perfil del mesero sólo se puede abrir cuentas nuevas, adicionar pedidos a las mesas existentes e imprimir los tickets del total de la cuenta, sólo puede visualizar los pedidos de sus propias mesas, no puede acceder a la información de los demás meseros.

Otro ejemplo es el perfil del cajero, el cual puede acceder a la información de todos los meseros que se generó durante el día, para saber únicamente y con exactitud el importe que se le debe cobrar al cliente.

Por último, en el perfil del administrador es quien puede acceder a toda la información disponible en el sistema, puede realizar altas, bajas o cambios en la información.

- ***Manipular en poco tiempo grandes cantidades de información***

  Esto es de gran utilidad sobre todo si se requiere conocer la cantidad de clientes que visita el negocio a cierta hora del día, saber con exactitud lo que ha consumido una mesa y la cantidad total que se le va cobrar al cliente, obtener un resumen de cuales mesas ya fueron cobradas y cuántas mesas quedan pendientes por cobrar, aplicar descuentos o promociones a un cierto rango de productos o a un grupo clientes, entre otros factores.

- ***Gestión de inventarios***

  Permite garantizar la rotación completa de toda la mercancía existente y evitar que los productos perecederos lleguen a su fecha de caducidad, saber el valor total de la mercancía que se encuentra en el almacén, determinar cuánta mercancía se consume por día, tener un detalle de los proveedores y saber con precisión cuánta mercancía solicitar a los proveedores y evitar excedentes.

- ***Interacción con software de Contabilidad Financiera***

  Delegar la tarea de las operaciones contables a software especializado en temas financieros y fiscales, únicamente servir como fuente de datos para aplicaciones externas, mediante interconexiones a estas plataformas.

- ***Almacén y análisis de datos***

  Con toda la información recopilada y almacenada en el sistema se puede aplicar análisis estadístico o minería de datos, para conocer a detalle el comportamiento de los clientes, sus preferencias de consumo, la mercancía que se debe solicitar anticipadamente a los proveedores con base al consumo de los clientes en días determinados, etc.

- ***Hardware especializado***

  En cuanto a hardware, el sistema de punto de venta está conformado básicamente por pantallas táctiles, impresora de tickets, computadora central, caja

registradora, lector de tarjetas bancarias y scanner de código de barras. Sin embargo, en la mayoría de los casos, no se requiere de hardware especializado o de una marca de hardware específica. El sistema debe tener la capacidad de adaptarse a cualquier implementación de hardware.

1.3 Principales componentes de hardware y software empleados en puntos de venta de restaurantes

En este apartado se realiza una breve investigación sobre algunas marcas comerciales de software de punto de venta que existen. Se realiza un énfasis en algunos puntos fundamentales.

- Descripción breve del sistema.
- Requisitos de hardware.
- Alcance de la versión (cuantos equipos soporta la licencia, es local o en la nube, cargos extra, soporte, etc.).
- Vigencia de la licencia.
- Costo total de la licencia.

1.3.1 Software de puntos de venta para restaurantes

Para obtener un estimado del costo total del sistema, no se tomaron en cuenta factores como soporte técnico, usuarios adicionales, reportes extras, modificaciones al software de caja, debido a que este factor representa un costo adicional variable. También se excluye de esta comparativa el costo de los timbres fiscales y comisiones para pagos con tarjeta. La fecha en que se realizó la investigación corresponde al mes de marzo del 2018.

Los sistemas funcionan de manera local y haciendo uso de los servicios del cómputo en la nube para sincronizar la información y generar reportes desde cualquier explorador web o dispositivos con acceso a internet, la vigencia de la licencia es permanente si se realiza un pago único, aunque se limita el software de algunas funcionalidades en comparación al pago mensual.

Entre algunas marcas comerciales de software, se contempla investigar los siguientes sistemas:

- Soft Restaurant
- EasyPOS
- Anova
- POSTER

1.3.1.1 Soft Restaurant

- **Descripción breve y sus características**

Soft Restaurant[2], es un software desarrollado por la empresa National Soft®, maneja una excelente interfaz para poder llevar acabo su operación en el punto de venta con los diferentes tipos de servicios: comida rápida, comedor, servicio a domicilio y drive thru[3] , contando con opciones y funciones como:

- Cancelaciones
- Juntar cuentas
- Dividir cuentas
- Cambiar cuentas
- Multi-impresión
- Descuentos
- Promociones
- Mapa de mesas

- **Requisitos del sistema**

En la tabla 1.1 se muestra la información sobre los requisitos del sistema, especificados en la página web del software, para el correcto funcionamiento del servidor.

[2] www.nationalsoft.store
[3] Negocio de comida, donde el cliente solicita el servicio sin salir de su automóvil.

Servidor

Requerimiento	Mínimo	Recomendado
Espacio disponible en Disco Duro	10 GB	40 GB
Procesador	Intel/AMD 32 bits con 1 núcleo a 1.6 GHz	Intel/AMD 32 bits con 4 núcleos a 2.6 GHz o superior
Memoria	2 GB	4 GB
Monitor	1024 x 768	1024 x 768
Sistema Operativo	Windows 8 PRO / Windows 10 Home / Windows 10 PRO	Windows 8 PRO / Windows 10 Home / Windows 10 PRO / Windows Server 2008 R2

Tabla 1.1 Requisitos para el servidor de Soft Restaurant

Terminales Punto de Venta

En la tabla 1.2 se presenta la información referente a los requisitos para las terminales o dispositivos de acceso al sistema de punto de venta.

Requerimiento	Mínimo	Recomendado
Espacio disponible en Disco duro	5 GB	10 GB
Procesador	Intel/AMD 32 bits con 1 núcleo a 1.6 GHz	Intel/AMD 32 bits con 2 núcleos a 1.6 GHz o superior
Memoria	1 GB	2 GB
Monitor	1024 x 768	1024 x 768
Sistema Operativo	Windows 8 PRO / Windows 10 Home / Windows 10 PRO	Windows 8 PRO / Windows 10 Home / Windows 10 PRO / Windows Server 2008 R2

Tabla 1.2 Requisitos para las terminales punto de venta de Soft Restaurant

- **Costo Licencia**

En la tabla 1.3 se muestra el costo de la licencia.

Módulo	Precio
Versión Profesional Pago Único	$ 11,600 MXN
Versión Profesional Pago Mensual	$ 870 MXN
Versión Estándar Pago Único	$ 6,960 MXN
Versión Estándar Pago Mensual	$ 522 MXN
Monitor Ventas Pago Único	$ 6,960 MXN
Monitor Cocina Pago Mensual	$ 580 MXN
Reservaciones Pago Mensual	$ 580 MXN

Tabla 1.3 Costo de la licencia de Soft Restaurant

1.3.1.2 EasyPOS

- **Descripción breve y sus características**

EasyPOS[4], es un software desarrollado por la empresa del mismo nombre EasyPOS®, donde los meseros pueden abrir y atender mesas directamente desde un mapa personalizado, levantar comandas fácilmente, mientras el cajero lleva el control de los consumos y pagos. Monitorear en tiempo real el negocio. Personalizar los tickets de venta, administrar el catálogo de productos y menú.

El sistema dispone de las siguientes características para mejorar el control de la administración:

- Crear usuarios con diferentes perfiles de accesos y permisos para cada área de negocio.
- Configurar productos por categoría y subcategoría.
- Generar reportes para mostrar cancelaciones, descuentos, etc., y exportar los reportes a Excel y PDF.
- Controlar las cuentas abiertas.
- Aceptar múltiples formas de pago y realiza cortes de caja.

[4] http://landing.easypos.com.mx/Informes

- Ingresar impuestos a cada producto.
- Emitir facturas electrónicas CFDI a clientes en México.

- **Requisitos del sistema**

  En la tabla 1.4 se muestran los requisitos del sistema, especificados en la página web del software.

Requerimiento	Recomendado
Sistema operativo	Windows 7 SP1 en adelante (no RT)
Procesador	Intel Atom en adelante
Memoria RAM	2 GB
Espacio en disco duro	5 GB
Monitor	Resolución mínima 1024x768

Tabla 1.4 Requisitos de EasyPOS

- **Costo Licencia**

  En la tabla 1.5 se muestra el costo de la licencia.

Módulo	Precio
easyPOS Servicio de Mesas Pago Único	$ 3,190 MXN
Inventarios Pago Único	$ 1,920 MXN
Cocina Pago Único	$ 640 MXN
Tablero de Control Pago Único	$ 1,920 MXN

Tabla 1.5 Costo de la licencia de EasyPOS

1.3.1.3 Anova

- **Descripción breve y sus características**

Anova[5], es un software inteligente de punto de venta para restaurantes, bares y cafeterías desarrollado por la empresa del mismo nombre Anova®. Es útil para inventarios, emite mensajes de alerta cuando es buen momento para comprar insumos, gracias a sus algoritmos inteligentes. También se puede generar perfiles personalizados de acuerdo a cada tipo usuario, asignar permisos y restricciones a las personas que utilizan el sistema.

En el sistema se puede realizar corte de caja, para que al final del día se registre el dinero. Anova tiene la capacidad para decir si sobra o hace falta dinero. Una de sus mejores características del sistema es el mapa de mesas. Toda la información sobre las mesas se muestra en una sola pantalla para hacer las operaciones más rápidas, desde dividir una mesa hasta pagar una cuenta. Al capturar los pedidos, sus tickets se imprimen inmediatamente en la barra de bebidas y en la cocina. Tiene la facilidad de agrupar los productos por familias y categorías.

- **Requisitos del sistema**

En la tabla 1.6 se muestran los requisitos del sistema, especificados en la página web del software.

Requerimiento	Recomendado
Sistema operativo	Windows 8 en adelante

Tabla 1.6 Requisitos de Anova

- **Costo Licencia**

En la tabla 1.7 se muestra el costo de la licencia.

Módulo	Precio
Pago Único	$ 7,499 MXN
Pago Mensual	$ 450 MXN

Tabla 1.7 Costo de la licencia de Anova

[5] http://anova.mx/index.html

1.3.1.4 POSTER

- **Descripción breve y sus características**

Poster[6], es un software desarrollado por la empresa Poster POS Inc. ®, dicha aplicación es accedida desde la nube y está enfocada a puntos de venta y administración de inventario para cafeterías, restaurantes y tiendas, el sistema POS, combina soluciones para front-office, inventarios, finanzas, análisis y CRM.

La aplicación se puede ejecutar en tablets con sistema operativo iOS, Android o Windows.

Es posible realizar la administración del negocio desde cualquier parte del mundo. La consola de administración de POSTER es accesible desde el explorador web por medio de un login y password desde cualquier dispositivo con acceso a internet, por lo que se puede revisar las ventas y administrar el negocio.

Con el sistema se puede agregar rápidamente suministros: atajos convenientes en el teclado, separar los módulos para iOS y aplicaciones Android. Calcular automáticamente los costos por producto, así como la utilidad neta por cada uno de ellos. Se muestran mensajes de alerta para mejor el control del inventario y no olivar comprar los productos a tiempo.

Con el software POSTER se ve en tiempo real, la cantidad de productos que se encuentran en el almacén. Además, se puede configurar un número límite de existencias bajas en el almacén para cada uno de los ingredientes y cuando un ingrediente rebasa este límite se generan alertas automáticas notificando al usuario.

[6] https://joinposter.com/en

- **Requisitos del sistema**

En la tabla 1.8 se muestran los requisitos del sistema, especificados en la página web del software.

Requerimiento	Recomendado
Sistema operativo	Windows 8.1 en adelante Android 4.4 en adelante iOS 10 en adelante
Procesador	Windows - 1.8 Ghz
Memoria RAM	Windows - 4 GB Android - 2 GB
Monitor	Android - 1024x600

Tabla 1.8 Requisitos de POSTER

- **Costo Licencia**

  En la tabla 1.9 se muestra el costo de la licencia.

Módulo	Precio
Versión Startup Pago Mensual	$ 24 USD
Versión Mini Pago Mensual	$ 34 USD
Versión Business Pago Mensual	$ 44 USD
Versión Pro Pago Mensual	$ 64 USD

Tabla 1.9 Costo de la licencia de POSTER

1.3.2 Hardware para puntos de venta restaurantes

Según las especificaciones de los proveedores de software, no se requiere adquirir hardware específico para que funcione cada uno de los sistemas vistos anteriormente, sin embargo, en la siguiente sección se realiza una comparativa sobre el costo de las diferentes marcas de hardware especializado para puntos de ventas.

Se excluye la información referente a equipos de cómputo y tablets, ya que el costo es muy variado y depende de las capacidades de los equipos.

1.3.2.1 Pantallas táctiles

En la tabla 1.10 se realiza una comparativa entre algunas marcas comerciales de pantallas táctiles.

Marca/Modelo	Descripción	Precio
NS Tech POS Touch 17”	Equipo POS touch screen de 17” disco duro de estado sólido de 64 GB SSD MSATA y memoria RAM de 4 GB	$ 899 USD
EC-TS-1510 15”	Monitor touch screen especial, uso rudo, LED, EC Line, 15 pulgadas, EC-TS-1510, TFT, POS	$ 7,196 MXN
Elo TouchSystems 15.6"	Sistema POS 15.6", Intel Celeron J1800 2.41GHz, 2GB, 320GB	$ 16,099 MXN
LG 17MB15T 17”	Monitor touch screen Resolución 1280 x 1024 Pixeles. Montable a la pared.	$ 6,599 MXN

Tabla 1.10 Costo de las pantallas touch más comunes

1.3.2.2 Impresora de tickets

En la tabla 1.11 se realiza una comparativa entre algunas marcas comerciales de impresoras de tickets.

Marca/Modelo	Descripción	Precio
Epson TM-T88V	Impresora de Tickets, Térmica Directa, Paralelo + USB, Negro	$ 6,632 MXN
Qian Anjet 80	Impresora de Tickets, Línea Térmica, 203 x 203DPI	$ 2,399 MXN
Epson TM-T20II	Impresora de Tickets, Térmico, Alámbrico, Serial + USB	$ 3,449 MXN
EC Line EC-PM-80330	Impresora de Etiquetas, Térmica Directa, Alámbrico, 203 x 203DPI	$ 3,129 MXN

Tabla 1.11 Costo de las impresoras de tickets más comunes

1.3.2.3 Gabinete de valores

En la tabla 1.12 se realiza una comparativa entre algunas marcas comerciales para guardar valores.

Marca/Modelo	Descripción	Precio
Subarasi MB1000	Compartimientos de monedas 8, Compartimientos de billetes 4	$ 679 MXN
EC Line EC-CD-50M	Cajón de Dinero EC Line con Llave, 5kg, Negro	$ 819 MXN
POSline CD030	Charola Removible para Cajón de Dinero, 8kg, Negro	$ 549 MXN
ZW CAJ404	Cajón De Dinero Punto De Venta Conexión Miniprinter Rj11	$ 749 MXN

Tabla 1.12 Costo de los gabinetes de valores más comunes

1.4 Planteamiento del problema

Durante esta breve investigación, se indagó sobre los elementos básicos que integran a un sistema de punto de venta para restaurantes, considerando aspectos de software y hardware.

Continuando con el análisis, implementar un sistema de Punto de Venta requiere de una fuerte inversión, aunque aparentemente no es demasiado costoso, en particular los costos del software. En muchos de los casos se limitan las funcionalidades completas que incluye el sistema, de modo que los proveedores de software obligan al cliente a comprar módulos adicionales o pagar por servicios extras, soporte técnico, capacitaciones, cantidad de usuarios, reportes adicionales, etc.

Además del costo que implica la compra de software, se tiene que incluir el costo del hardware. Entre más robusto sea el sistema, se requiere adquirir más dispositivos y

equipos de cómputo que sean capaces de soportar la operación y el procesamiento de información.

En la mayoría de los casos, el hardware especializado para sistemas de puntos de venta, no es necesario, sin embargo se suele utilizar para mayor comodidad de los usuarios (meseros, cajeros, administradores, gerentes), un ejemplo de ello son las pantallas táctiles, que resultan muy prácticas de utilizar, donde el usuario se adapta fácilmente a ellas, ya que el área para visualizar los datos es mayor, en comparación a una tablet o un teléfono celular, pero éste es uno de los dispositivos que resulta ser muy costoso.

Al tratarse de software propietario y de pago, difícilmente el cliente pude realizar modificaciones al sistema de caja, adecuarlo conforme a sus necesidades y modificar el código fuente, incluso aunque se realice el pago por alguna modificación, se tiene que tener cuidado sobre la integridad de los datos, y conservar las reglas de negocio ya establecidas.

Por ello no todas las PyMES tienen la capacidad financiera de adquirir este tipo de aplicaciones y solventar los costos post-instalación que conlleva este proceso, ni tampoco están interesados en complicar sus procesos, agregando funciones extras a su negocio.

Partiendo de los puntos anteriormente descritos, surge la necesidad de implementar un software simple que ayude a la administración del negocio, teniendo como principales requisitos:

- Bajo costo
- Escalabilidad en hardware y software
- Portabilidad multiplataforma
- Acceso a la aplicación desde diferentes dispositivos
- Manejo de roles para los usuarios
- Respuesta en tiempo real
- Intuitivo y fácil de usar para los usuarios

Los esfuerzos realizados en este trabajo de tesis, están orientados a diseñar y desarrollar una solución a la problemática planteada, implementando tecnologías nuevas y que están en auge de crecimiento, como lo son, los frameworks asíncronos, donde se reduce el tiempo de respuesta y no hay bloqueos durante la ejecución de la aplicación.

Entre las herramientas propuestas para dar solución a la problemática, se encuentran las bases de datos NoSQL, que representan una alternativa para almacenar y

manipular datos, siguen otro tipo de enfoque como el manejo de esquemas dinámicos, escalamiento y distribución de la información entre otros.

El enfoque tradicional es utilizar bases de datos relacionales que son muy útiles para el tema de transacciones y consistencia de los datos, sin embargo, tienen dificultades para el manejo de datos no estructurados.

En capítulos posteriores se realizará un análisis para determinar en qué casos es conveniente ocupar un tipo de base de datos y para que otros casos no son la mejor solución.

Capítulo 2

Panorama general, frameworks asíncronos y bases de datos NoSQL

2. Panorama general, frameworks asíncronos y bases de datos NoSQL

2.1 Frameworks asíncronos

Antes de entrar en contexto para saber, que son los Frameworks asíncronos, es importante entender y comprender los principales conceptos en cuanto a modelos de programación síncrona y asíncrona.

2.1.1 Modelo de programación síncrona

Este modelo de ejecución se utiliza comúnmente en la mayoría de las aplicaciones. Consiste en una ejecución secuencial de cada uno de los módulos del programa, se ejecuta uno tras otro y no se ejecutan los siguientes módulos hasta que se haya completado el módulo anterior. Las actividades se ejecutan en un solo thread (hilo de ejecución o subproceso).

La desventaja de este tipo de programación es el bloqueo de los módulos del programa. Si un fragmento de código no finaliza correctamente, no se puede avanzar a la siguiente actividad y toda la aplicación quedará bloqueada.

Uno de los ejemplos más sencillos que podemos encontrar es el modelo de ejecución síncrona sobre un solo hilo de ejecución, como se muestra en el ejemplo de la figura 2.1.

Figura 2.1 Diagrama de ejecución síncrona sobre un solo hilo

Existe una variante dentro del modelo de ejecución síncrona: Programación multi-hilos. En este modelo, cada hilo es manejado por el sistema operativo, el cual se encarga de asignar los recursos necesarios para cada uno de ellos, como procesador, memoria, escritura en disco, en general, operaciones de entrada y salida, como se ejemplifica en la figura 2.2.

Figura 2.2 Diagrama de ejecución síncrona multi-hilo

El modelo de programación síncrona multi-hilo, tiene la ventaja que mientras un hilo de ejecución se encuentra bloqueado, los demás hilos pueden continuar ejecutando sus procesos. Mientras tanto, entre las desventajas que podemos encontrar al utilizar la programación multi-hilo son el difícil manejo y administración de cada hilo. Se requiere un alto nivel de conocimientos y técnicas para realizar una correcta implementación y evitar problemas con el manejo de recursos del sistema y con la información con que se está procesando, en especial, el control de concurrencia para evitar corrupción de datos.

2.1.2 Modelo de programación asíncrona

Este tipo de programación se caracteriza por evitar bloqueos durante la ejecución de un programa o flujo de instrucciones sin la necesidad de emplear subprocesos o hilos de ejecución adicionales. Funciona sobre un solo hilo de ejecución y un modelo de asignación asíncrona de tareas en segundo plano (background).

Instrucciones o tareas que requieren hacer uso de recursos como son acceso a disco, acceso a recursos de red, bases de datos, etc., se les conoce como operaciones de Entrada/Salida (E/S). Estas operaciones requieren cierto tiempo para poder completarse y por lo tanto provocan que el flujo principal de ejecución se detenga o se suspenda. La idea en este modelo de programación es evitar estos “tiempos muertos” delegando su ejecución a otros componentes de los cuales se hablará más adelante.

La estrategia anterior permite que las peticiones que se realicen por parte de un actor, cliente o sistema sean prácticamente instantáneas ya que son peticiones libres de bloqueos.

Una vez que la tarea ha sido delegada para su ejecución en segundo plano, esta es procesada. Al terminar su ejecución, se notifica su conclusión y se presentan los resultados obtenidos.

Al evitar bloqueos en el flujo de ejecución de un programa, se obtiene como beneficio la posibilidad de recibir y procesar nuevas peticiones de tareas para ejecutar, ofreciendo un nivel de disponibilidad mucho mayor.

A nivel técnico, un concepto fundamental para implementar esta técnica de programación son las funciones o llamadas de retorno también conocidas como “callbacks”, cuyo propósito es retomar y terminar de ejecutar la tarea que se encontraba ejecutando en segundo plano. Dependiendo el resultado de la ejecución en segundo plano, se realiza la ejecución de esta función o callback.

Otra forma de visualizar este modelo es el asociar cierta lógica o código encapsulado en una función a una operación E/S. Esta función (callback) será ejecutada una vez que la operación E/S concluya. Generalmente el código de esta función realiza las acciones necesarias para concluir con la tarea originalmente asignada.

En la figura 2.3 se muestra este tipo de programación, la forma en la que se intercalan tareas, lanzando y retomando dichas tareas para finalizar su ejecución.

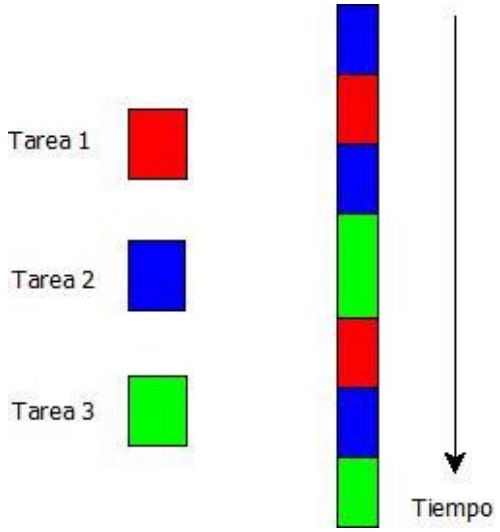


Figura 2.3 Diagrama de flujo de ejecución asíncrona

Visto desde una forma simple, este modelo de programación puede compararse con el estilo de trabajo de un mesero en un Restaurante. Mientras la mesa 1 revisa la carta y decide que ordenar, el mesero puede acudir a la cocina para obtener la orden de la mesa 2. Si en cocina la orden aún no está lista, el mesero no espera, continúa atendiendo otras mesas y posteriormente regresará para retomar la tarea que dejó en ejecución la cual fue delegada al área de cocina.

Este estilo permite maximizar el uso del tiempo sin la necesidad de hacer uso de varios meseros para atender a una mayor cantidad de clientes.

2.1.3 Definición de framework asíncrono y ejemplos

Un Framework es un conjunto de herramientas de software que nos ayuda a desarrollar aplicaciones complejas, dispone de módulos y librerías para resolver problemas específicos, como, por ejemplo, conexiones y consultas a base de datos, solicitudes y respuestas HTTP, interfaz gráfica, etc.

Son indispensables para evitar re-escribir código, además, el desarrollo se implementa con mejores prácticas, mayor seguridad, el código es más limpio, ordenado y mantenible.

No es conveniente utilizar un Framework en aplicaciones sencillas, debido a que tardaríamos más en configurar y programar el Framework que en utilizar el API estándar de un lenguaje de programación, además de que se consumen recursos innecesariamente.

Un Framework Asíncrono se caracteriza por utilizar el modelo de programación asíncrona. En general sólo utiliza un solo hilo de procesamiento para ejecutar un bucle de eventos, que se encarga de inicializar, monitorear y notificar el resultado del procesamiento de los eventos.

Un bucle lo podemos definir como una repetición de instrucciones que se ejecutan hasta que se cumple una condición. Un bucle de eventos es un conjunto de tareas que se van ejecutando hasta que se finaliza su ejecución. Estas tareas se acumulan en una cola y se atienden en el orden en el que van llegando de forma síncrona. Cuando se termina de ejecutar la tarea se envía una notificación al usuario sobre el resultado.

Hoy en día existe una gran variedad de Frameworks que emplean un paradigma asíncrono y prácticamente existen para todos y cada uno de los lenguajes de programación más comunes, como JavaScript, Java, Python y .NET.

En el siguiente apartado se mencionan las principales características de estos frameworks. Cabe mencionar que algunos de ellos fueron creados desde sus inicios específicamente para trabajar de forma asíncrona. En cambio, existen otros leguajes como Java que están incorporando módulos para poder trabajar en este tipo de programación asíncrona, empleando el término "programación reactiva". Más adelante se explicará sobre esta nueva forma de programación con Java.

2.1.3.1 Node.js

Es un framework asíncrono que funciona por medio de eventos, está enfocado para desarrollar aplicaciones de red que funcionan sobre los protocolos HTTP, streaming y sistemas escalables en JavaScript.

La mayoría de los sistemas en red que son concurrentes y se administran por medio de hilos, son relativamente ineficientes, complicados de administrar y dar mantenimiento al código. Comúnmente estas aplicaciones utilizan un hilo de ejecución por cada conexión que se realice, lo que ocasiona un incremento en el consumo de recursos, así como la liberación de los recursos al término de la conexión.

Una ventaja de Node.js es que los desarrolladores no tienen que preocuparse por el bloqueo de los procesos, debido a que éstos no ocurren, lo que permite a los desarrolladores enfocarse únicamente en la lógica del sistema. Las operaciones de Entrada/Salida (E/S) son delegadas al sistema operativo, a la base de datos o a los dispositivos y periféricos, mediante un nuevo subproceso. Node.js no tiene que actuar directamente sobre estas operaciones.

La ejecución en Node.js se realiza a través un solo hilo de ejecución. Cada vez que se realiza una solicitud al sistema, Node.js hace uso de la librería llamada `libuv`, la cual es responsable de atender todas las operaciones asíncronas dentro del loop de eventos. Tiene su propio entorno multi-hilos, por defecto utiliza cuatro hilos, las solicitudes se van encolando y son atendidas mientras existan hilos de ejecución disponibles.

Al finalizar la tarea, también son manejadas por la librería `libuv`, quien realiza la ejecución de las llamadas de retorno (callbacks).

Características de Node.js

El propósito general de Node.js es atender múltiples solicitudes HTTP, con un solo hilo de ejecución, de forma asíncrona y no bloqueantes.

La secuencia para atender y responder solicitudes se muestra en la figura 2.4

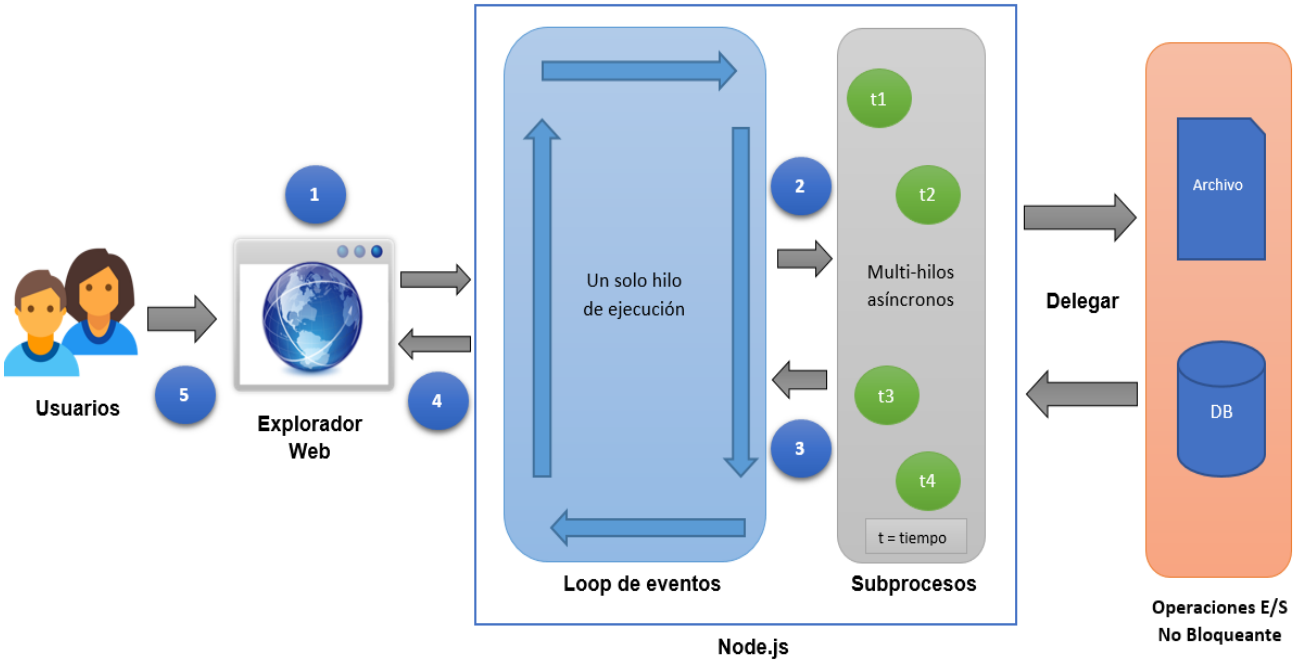


Figura 2.4 Diagrama de solicitudes Node.js

A nivel general se tiene la siguiente secuencia de eventos:

1. Se realiza una solicitud HTTP desde algún explorador web
2. Dependiendo de la solicitud, si es una operación de E/S, por ejemplo, una consulta a la base de datos, Node.js genera un nuevo subproceso y asigna la solicitud a la base de datos, para que esta realice las operaciones solicitadas, mientras que Node.js continúa atendiendo otras solicitudes.
3. Mientras se termina de ejecutar la operación de E/S, Node.js ya está preparando alguna plantilla HTML para mostrar los datos resultantes de la operación.
4. Node.js hace uso del loop de eventos, quien se encarga de procesar todos los eventos asíncronos y de recibir las notificaciones cuando estos finalicen. Ya con la información

obtenida de la base de datos, junto con la plantilla HTML asignada, Node.js prepara la respuesta para visualizar los datos obtenidos de la operación de base de datos en la página web.

5. Mientras tanto, otras solicitudes son atendidas dentro del mismo bucle de eventos.

Otra de las ventajas de usar Node.js, es la gran flexibilidad de trabajar con archivos de formato JSON con ayuda de Express, el cual es un framework que ofrece a los desarrolladores herramientas robustas para aplicaciones web y móviles. En conjunto hacen más fácil y amigable el desarrollo de aplicaciones complejas.

Incluso no solamente se utiliza Node.js para el desarrollo web, sino también como un servidor de videojuegos, gracias a su eficiente desempeño en la implementación de sockets por medio de los protocolos TCP/IP, se puede construir un servidor para la transmisión de datos sobre el estado del videojuego a varios jugadores mientras éstos interactúan.

Otro factor que ayuda en gran medida al desarrollo de servidores de videojuegos sobre esta plataforma es la alta escalabilidad, ya que tiene la posibilidad de escalar a varios servidores y/o procesadores.

Una característica fundamental de Node.js es que utiliza el lenguaje JavaScript, comúnmente empleado del lado del cliente para el desarrollo de páginas web dinámicas, pero gracias a la empresa Google, que liberó el código fuente de su motor V8 de Google Chrome escrito en JavaScript, se logró implementar esta versión sobre Node.js.

Entre las principales características del lenguaje JavaScript se encuentran:

- Es un lenguaje interpretado: Es más rápido escribir y probar el código fuente, en comparación a los lenguajes compilados, al requerir de un proceso de compilación y generación de código ejecutable antes de iniciar y probar la aplicación. Sin embargo, una vez que es compilado, su ejecución es más rápida al haber sido traducido a un lenguaje de ejecución nativo (lenguaje máquina). Realmente el lenguaje interpretado hace más rápido el desarrollo y portable, aunque esto implique un costo en tiempo de ejecución.
- La programación es funcional, los estados de las variables son inmutables (en funciones puras), las funciones reciben datos como parámetros que son procesados y devueltos,

por lo que el resultado es predecible, su enfoque se centra en ¿Qué? y no en el ¿Cómo? se realizan las actividades.

- Orientado a objetos y eventos: no es 100 % orientado a objetos ya que no posee las características como tal de este tipo de programación, sino que en lugar de manejar clases se utilizan prototipos.

Arquitectura de Node.js

Node.js está compuesto de una librería estándar, módulos del sistema y NPM que es el manejador de paquetes de JavaScript del inglés (Package Manager for JavaScript). Su librería estándar está dividida en dos partes, librerías de binarios entre las cuales se incluye `libuv`, la cual administra los eventos para que no queden bloqueados y se encarga de las llamadas de E/S, también incluye una librería dedica a los servicios HTTP. La otra parte de la librería estándar son los módulos del Core. En la figura 2.5 se puede apreciar más a detalle esta descripción.

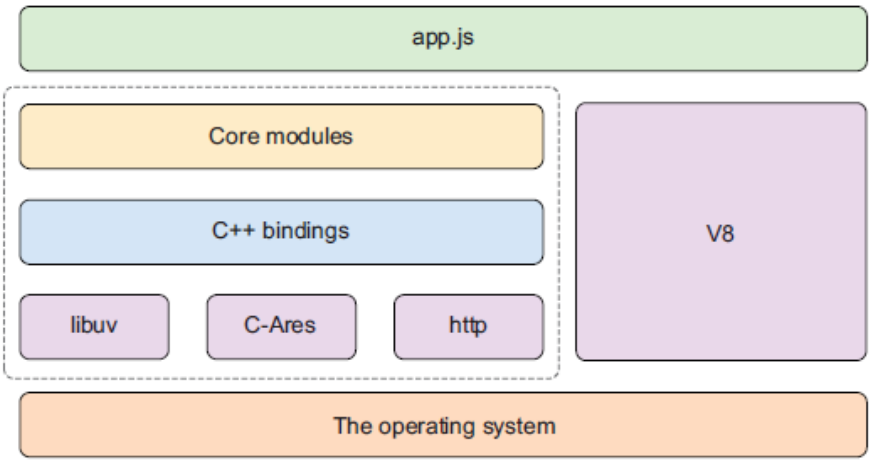


Figura 2.5 Arquitectura Node.js

Módulos Node.js

Entre algunos de los módulos más importantes de Node.js podemos encontrar:

- **EventEmitter**: es el módulo principal de Node.js, de éste se derivan otros módulos como los de red, los sistemas de archivos y streams.
- **Stream**: hereda de `EventEmitter`, con este módulo se crea un objeto que recibe eventos de una conexión, es útil para la trasferencia de datos, a través de un flujo

de datos podemos saber cuándo ha terminado la trasferencia y en caso de surgir algún error nos informe sobre éstos.

- **Fs**: hace referencia al sistema de archivos, podemos leer y escribir archivos partiendo de los principios de no bloqueo
- **Net**: es la base para desarrollar clientes y servidores asíncronos con el protocolo HTTP.
- **NPM**: como tal no forma parte dentro de los módulos funcionales de Node.js, más bien, es un módulo externo a Node.js el cual provee de software y administra paquetes escritos en JavaScript, desde un repositorio se extraen los paquetes que son escritos por una gran cantidad de desarrolladores que realizan aportes por medio de la comunidad npm orgs, esta herramienta se instala junto con Node.js.

El diseño Node.js se asemeja a otros sistemas que implementan la misma lógica de programación, como son, la máquina de eventos de Ruby cuyas características es la alta escalabilidad, estabilidad y eficiente rendimiento para ambientes productivos altamente demandantes.

Otro Framework muy parecido es Twisted de Python, su motor está enfocado a manejo de eventos de aplicaciones en red, está bajo licencia MIT. Entre sus características, soporta múltiples capas de los protocolos de red, como TCP, UDP, SSL, SSH, IRC y FTP. Está integrado con un cliente y un servidor para todas las partes de sus protocolos, lo que facilita la configuración y el despliegue en producción.

Implementaciones de Node.js

Casos que podemos encontrar, donde se implementa Node.js en algunos módulos de los servicios, debido a su eficiente rendimiento para atender una gran cantidad de usuarios en tiempo real encontramos:

- https://www.paypal.com
- https://www.netflix.com
- https://www.uber.com
- https://www.linkedin.com
- https://www.ebay.com/

2.1.3.2 Reactor 3

Es una implementación del paradigma de programación Reactiva en Java, se integra directamente con las APIs de Java 8, específicamente con las clases `CompletableFuture`, `Stream` y `Duration` que están orientadas para tratar temas de computación asíncrona.

El modelo se enfoca en el flujo de datos y la propagación de los mismos. Al ser llamada programación reactiva, quiere decir que recibe un estímulo, este estimulo es el flujo de la transmisión de datos, llamado Streams.

Este tipo de programación lo podemos encontrar también en las librerías Reactive Extensions (Rx) bajo la plataforma .NET. Reactor 3 mejora de las versiones RxJava 1 y RxJava 2, en versiones más recientes de Java como la versión 9 se implementaron mejores características para el manejo de flujos.

La programación reactiva está basada en el patrón de diseño `Observer`, el cual nos permite realizar el monitoreo de un objeto si este cambia su estado al realizar ciertas acciones, de forma similar lo podemos comparar con el patrón de diseño `Iterator`, que implementa los mecanismos para acceder a los elementos de una colección de forma secuencial. La principal diferencia entre ambos patrones de diseño es que `Iterator` se basa en "`pull`" y la programación reactiva está basado en "`push`".

Una forma simple de explicar esto, es a través de la interface `Iterator`, en donde tenemos que actuar directamente sobre los objetos para cambiar el valor de los mismos, en cambio en el enfoque de la programación reactiva, se monitorea al objeto, si este cambia el valor de sus atributos, entonces se propaga el cambio a los demás objetos.

Las aplicaciones modernas por lo general alcanzan un número enorme de usuarios que utilizan el sistema al mismo tiempo, aprovechando el uso de hilos que se ejecutan de forma simultánea.

Sin embargo, al incrementar los hilos de ejecución se vuelve un problema para el desempeño de la aplicación, principalmente por el uso de recursos. En la mayoría de los casos, los programadores en Java escriben código síncrono. Lo que ocasiona que los recursos no sean aprovechados por otros hilos en ejecución y en el peor de los casos los recursos no pueden ser accedidos por que se encuentran bloqueados.

Una solución para evitar el desperdicio de recursos es escribir código asíncrono no bloqueante, sólo se encarga de ejecutar tareas activas dejando en background tareas no relevantes. Una vez que su ejecución concluye, se retorna la respuesta al módulo encargado de procesar la tarea.

En Java antes de la versión 8 existen dos modelos para implementar la programación asíncrona.

- **Callbacks**

Son métodos asíncronos que son llamados cuando el resultado de la ejecución está disponible, no regresan un valor, toman un parámetro en el que se envía la función callback a ejecutar. Por lo general son expresiones lambda o clases anónimas para realizar la llamada al código de ejecución que finaliza la tarea.

- **Futures**

Son métodos asíncronos que implementan la interface `Future<T>`, representa el resultado de una operación asíncrona, revisa constantemente si la operación ya está completa, sino espera hasta ser completada. Una vez completada la operación recupera el resultado. Con el método `get` se obtiene el resultado de la operación.

Características de Reactor 3

Esencialmente los sistemas reactivos que están enfocados a ofrecer servicios por medio de internet y que además requieren de una infraestructura de gran escala, deben cumplir con las siguientes características:

- **Responsivos**

El sistema debe responder de manera oportuna en todo lo posible, detecta rápidamente los problemas y son manejados efectivamente, responde a tiempos de respuesta establecidos en caso contrario son manejados como errores.

- **Resilientes**

Es un sistema que se mantiene receptivo ante la presencia de una falla. Cada tipo de falla es manejada dentro de cada componente de forma aislada y así es como se asegura que no se comprometa todo el sistema.

- **Elásticos**

Sin importar el aumento de la carga de trabajo el sistema se mantiene receptivo, aumentando o disminuyendo el uso de recursos asignados al servicio evitando cuellos de botella distribuyendo la carga de trabajo.

- **Manejo de mensajes**

Para desacoplar los componentes, la interacción entre los mismo es por medio de mensajes asíncronos.

Estas características fundamentales surgen de la necesidad de obtener un mejor desempeño en el uso del procesador ante las acciones de E/S y minimizar el uso de memoria debido al enorme crecimiento de uso de múltiples hilos por cada pool de conexión que se realiza. En la figura 2.6 se muestran los principios de la programación reactiva.

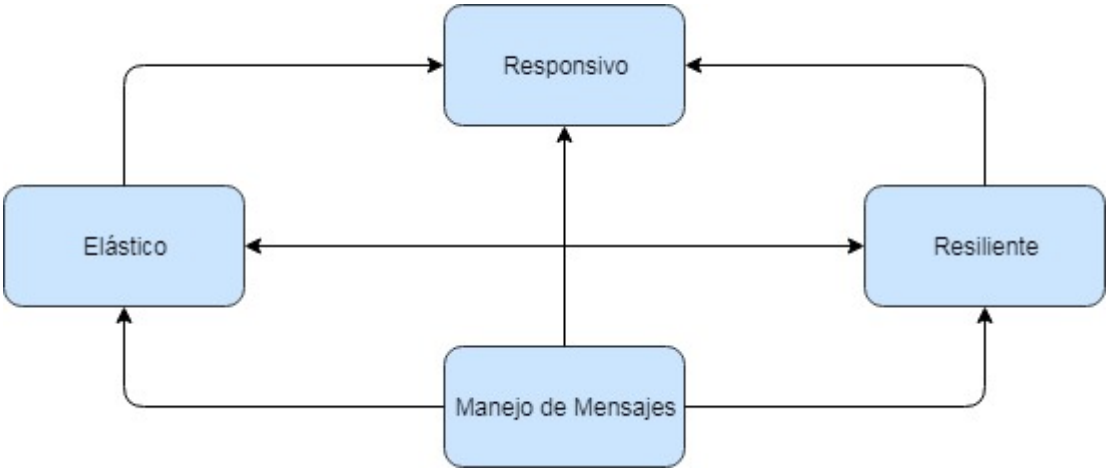


Figura 2.6 Principios de la programación Reactiva

Arquitectura de Reactor 3

Dentro del proyecto Reactor, el principal módulo es reactor-core, es una librería enfocada al manejo de flujos que existe desde la versión 8 de Java pero en la versión 9 se introdujo el módulo de `Reactive Streams` el cual posee mejores características que su versión anterior. En la figura 2.7 se muestran los nuevos módulos implementados en Java 9.

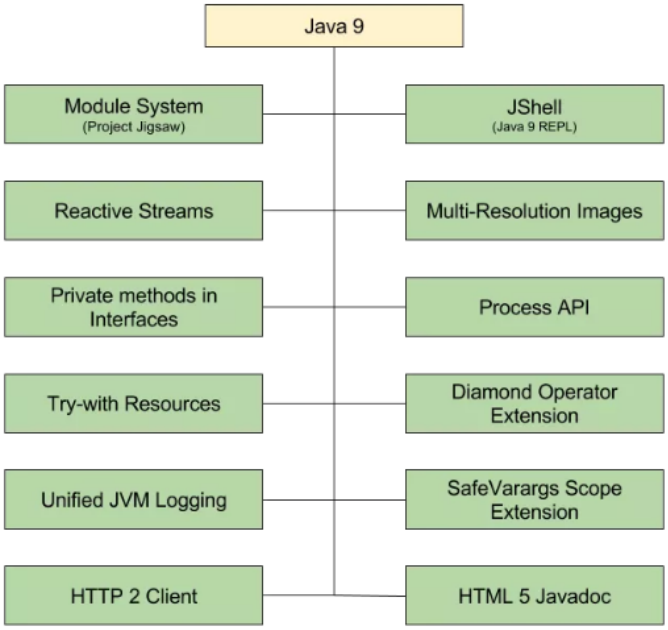


Figura 2.7 Módulos de Java 9

Bajo este contexto existen dos principales clases `Flux` y `Mono` que implementan a la interface `Publisher`.

Un objeto generado por la clase `Flux` representa una secuencia reactiva de 0 a N elementos. Mientras tanto un objeto que se origina de la clase `Mono` representa un resultado de 0 a 1

- **`Flux`**

Es un flujo reactivo asíncrono que implementa a `Publisher`, emiten de 0 a N elementos, las opciones de señal que se reciben son de un valor, completado o error, de forma análoga sus respectivos métodos son `onComplete` u `onError` y otro método que podemos encontrar es `onNext`, que se puede aplicar a cada elemento simple para realizar una acción. Como se observa en la figura 2.8

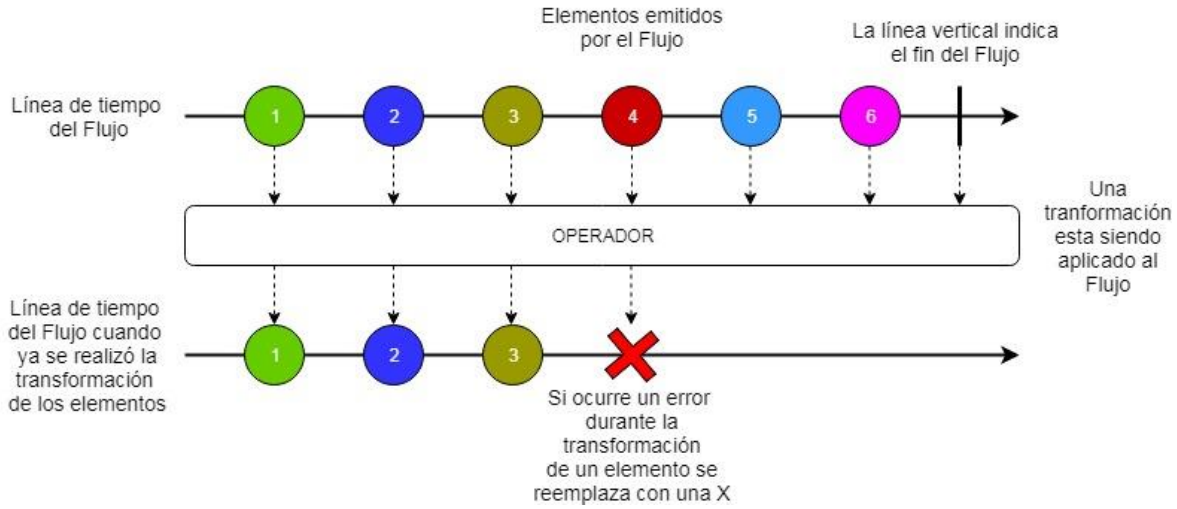


Figura 2.8 Diagrama de funcionamiento de Flux

- **Mono**

Es una implementación de `Publisher` que emite a lo más un elemento y termina opcionalmente con una señal de respuesta `onComplete` u `onError`.

Si se sabe que se emitirán de 0 a 1 elemento, debe emplearse la clase Mono, para tratar al elemento emitido, como se muestra en la figura 2.9.

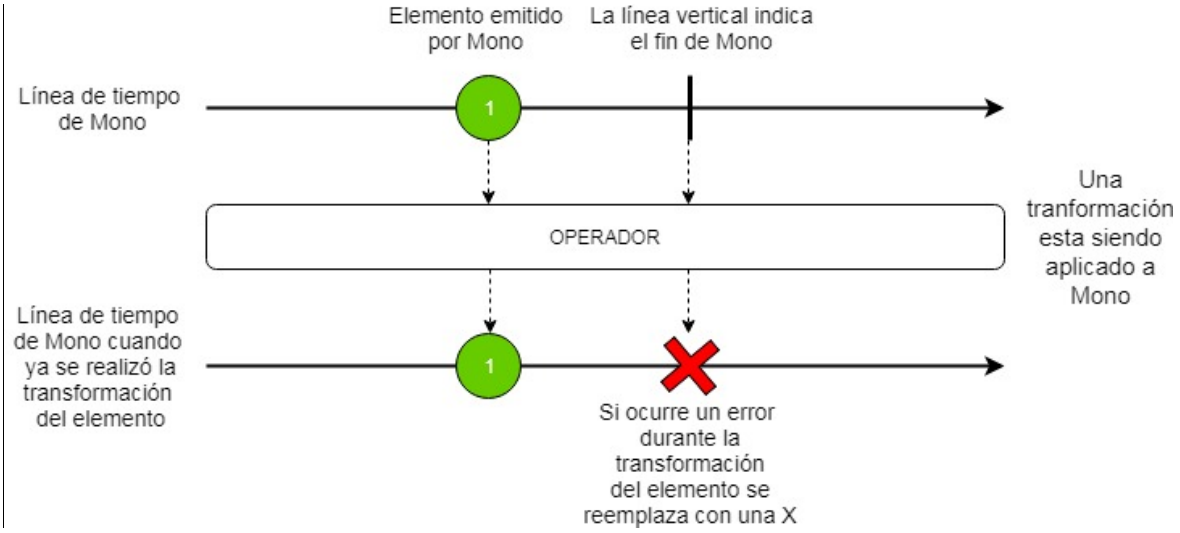


Figura 2.9 Diagrama de funcionamiento de Mono

Implementaciones de Reactor 3

Actualmente sólo podemos encontrar implementaciones de este tipo en aplicaciones móviles y servicios web.

- Aplicaciones Android
- Aplicaciones en Spring que implementen módulos asíncronos.
- Microservicios

2.1.3.3 .Net Framework

Dentro de la plataforma .NET desarrollada por la empresa Microsoft, existen tres patrones fundamentales para el manejo de procesos asíncronos llamados *Patrones de programación asíncronos.*

Patrones de programación asíncronos en .NET

A continuación, se exponen los diferentes tipos de patrones de programación asíncrona que existen en el framework .NET

- **Patrón asíncrono basado en tareas (TAP).**

Este patrón de basa en objetos Task (tareas) que caracterizan este patrón de programación. Usa métodos para saber cuándo inicia o finaliza una operación asíncrona, comúnmente a los métodos se les identifica con el sufijo `Async`.

El valor que puede retornar una operación asíncrona es de tipo `System.Threading.Tasks.Task<TResult>`.

Dentro de este tipo de programación en algunas ocasiones se realizan operaciones de tipo síncrono es decir que bloquean momentáneamente el proceso, algunas operaciones son como validar argumentos o ejecutar nuevas operaciones asíncronas, para después continuar con el retorno de los resultados una vez finalizada la operación asíncrona inicial. Por lo que, en la medida de lo posible, se tiene que minimizar este tipo de operaciones para no mermar el tiempo de respuesta de la aplicación.

Unos de las consideraciones que se deben tener presentes en este tipo de programación TAP para el retorno efectivo de los resultados son:

- Si los métodos asíncronos son invocados desde una interface de usuario hay que considerar que un prolongado tiempo de ejecución dañaría severamente el tiempo de respuesta de la aplicación.
- En caso de ejecutar varios métodos asíncronos concurrentes se tiene que considerar que dentro de estos métodos existen porciones de código que se ejecutan de forma síncrona, por lo que al sumar los tiempos que toma cada proceso síncrono, afecta al tiempo de respuesta y no se obtienen los beneficios esperados de la programación asíncrona.

Se tiene que valorar si es necesario realmente aplicar la ejecución de una operación asíncrona, ya que en algunos casos el esfuerzo y el costo computacional aplicado a una operación asíncrona son mayor que una operación síncrona.

Cuando se implementa un método TAP se puede elegir la distribución de carga de trabajo sobre un determinado proceso de E/S. Incluso dentro de un método TAP puede, no tener una ejecución asíncrona y únicamente retorna una respuesta del resultado de un suceso ocurrido en algún otro lado del sistema operativo, como por ejemplo la llegada de un flujo de datos encolados.

Las tareas tienen un ciclo de vida para las operaciones asíncronas y es representado por la enumeración `TaskStatus` que representa la etapa actual en la que se encuentra una tarea, las propiedades comúnmente utilizadas son: `IsCanceled`, `IsCompleted`, y `IsFaulted`.

Uno de los métodos que son de gran utilidad y que permite utilizar este patrón de programación es la cancelación de tareas, siempre y cuando la operación lo permita.

Otra de las características de este patrón de programación, es la notificación del progreso de la ejecución, éstos son típicamente utilizados para actualizar la interfaz de un usuario con información acerca del progreso de una operación asíncrona.

- **Patrón asíncrono basado en eventos (EAP).**

Para aplicaciones sofisticadas que requieren implementar un esquema multi hilos y que además se requiere un alto nivel de conocimientos para el correcto manejo de los subprocesos, una posible solución a esta problemática es el patrón asíncrono basado en eventos.

Este patrón asíncrono dispone de muchas ventajas para aplicaciones multi-hilo. Uno de los beneficios, es que oculta muchos problemas heredados del diseño de aplicaciones multi-hilo. Con lo que nos permite:

- Manejar y lanzar procesos en background sin interrumpir la ejecución de la aplicación, como los son, operaciones a bases de datos o descarga de archivos que contienen un gran volumen de información.
- Se puede ejecutar de forma simultánea múltiples operaciones y cuando se hayan finalizado las tareas se recibe una notificación.
- Tiene la capacidad de esperar por la disponibilidad de los recursos sin necesidad de detener la aplicación.
- Comunicación entre varias operaciones asíncronas por medio del modelo de eventos delegados.

Uno de los pilares en los que se fundamenta .NET es el modelo delegado, que se basa en el patrón de diseño `Observer` el cual consiste en habilitar un suscriptor para recibir notificaciones sobre algún evento ocurrido y definir las acciones que se deben realizar para responder a dicho evento.

Un evento es una señal de que ha ocurrido una acción, se envía en forma de mensaje por medio de un objeto, dicha acción es el resultado de un suceso como, por ejemplo, al oprimir un botón o cuando cambia el estado de un objeto, se puede considerar que ha ocurrido un evento.

Dentro de los objetos existen propiedades de tipo evento que sirve para invocar el evento y es conocido como event sender. El evento `Click` es miembro de la clase `Button` y el evento `PropertyChanged` es miembro de la clase que implementa la interface `INotifyPropertyChanged`.

Se conoce como delegado en .NET al objeto que sostiene una referencia a un método, son utilizados para pasar métodos como argumentos a otros métodos. Los controladores de eventos son métodos que se llaman por medio de un delegado. De esta forma se invoca de forma indirecta varias funciones cuando ocurre un evento.

Los delegados tienen muchos usos en .NET, en el contexto de eventos, un delegado es un intermediario entre el evento de origen y el código que se ejecuta cuando se lanza el

evento. Se asocia un evento con un delegado incluyendo un tipo de dato de retorno en una declaración de evento.

Los datos que están asociados con un evento pueden ser provisto mediante una clase de datos de eventos. El framework .NET provee de muchas clases de eventos que se pueden utilizar en las aplicaciones, tales como `SerialDataReceivedEventArgs` que proporciona datos del evento `DataReceived`.

La clase `EventArgs` es la base para todas las clases de eventos cuando se crea un evento que solamente notifica otras clases que algo ha ocurrido y no necesita pasar algún dato debe de incluirse como segundo parámetro la clase `EventArgs` en el delegado, también se puede pasar `EventArgs.Empty` cuando ningún dato es proporcionado.

Una clase que está basada en el patrón asíncrono de eventos, tendrá uno o más métodos llamados `NombreMetodoAsync` así mismo se implementan métodos para cuando se completa la operación o se cancela, `NombreMetodoCompleted` y `NombreMetodoAsyncCancel` respectivamente.

En caso de que se requiera que una aplicación que se mantenga funcionando mientras se está realizando una actualización en la base de datos, la cual tiene la característica de que esta operación consume mucho tiempo en realizar esta operación, se puede crear el método `UpdateAsync` y manejar el evento con el método `UpdateCompleted`.

En el momento en que se ejecuta el método `UpdateAsync` la aplicación continúa ejecutándose mientras se está realizando la actualización en la base de datos en un subproceso que se manda a background, el método del manejador de evento será llamado cuando se haya completado la operación en la base de datos.

El patrón asíncrono basado en eventos tiene la facultad de cancelar operaciones asíncronas, en la clase `PictureBox` se implementa el método `CancelAsync`, el cual envía una solicitud para detener el proceso pendiente y cuando el proceso es cancelado entonces es invocado el método `UpdateCompleted` para completar la operación.

- **Modelos de Programación Asíncrona (APM)**

La principal característica del modelo de programación asíncrona es que implementa la interface `IAsyncResult` y básicamente trabaja con dos métodos llamados

`BeginNombreOperacion` y `EndNombreOperacion` que inicializa y termina la operación `NombreOperacion`.

Cada vez que se ejecuta el método `BeginNombreOperacion` la aplicación principal puede continuar ejecutándose, mientras que la operación asíncrona continúa su ejecución sobre un subproceso alterno. Por cada llamada al método `BeginNombreOperacion`, la aplicación debería también llamar al método `EndNombreOperacion` para obtener el resultado de la operación.

El método `BeginNombreOperacion` comienza la operación asíncrona y retorna un objeto que implementa la interface `IAsyncResult`, la cual almacena información acerca de la operación asíncrona. Cuando se produce una excepción, el método `callback` no es invocado.

El método `EndNombreOperacion` finaliza la operación asíncrona, el valor de retorno es del mismo tipo retornado por su contraparte síncrona. Por ejemplo, el método `EndRead` retorna el número de bytes leídos de un `FileStream` y el método `EndGetHostByName` retorna un objeto `IPHostEntry` que contiene información acerca del host de la computadora.

Si la operación asíncrona representada por el objeto `IAsyncResult` no es completada cuando el método `EndNombreOperacion` es llamado, bloquea la llamada del proceso hasta que la operación asíncrona es completada.

Si se requiere intencionalmente que con la llamada al método `EndNombreOperacion` se bloquee el proceso hasta que uno o varios procesos que tengan dependencia con este proceso hayan finalizado, se utiliza la propiedad `AsyncWaitHandle` del objeto `IAsyncResult` retornado por la operación asíncrona del método `BeginNombreOperacion`.

Características de .NET Framework

Básicamente .NET fue hecho con la siguiente finalidad:

- Multiplataforma

Por medio del Common Languaje Runtime (CRL) quien se encarga de ejecutar las aplicaciones desarrolladas en .NET, se realiza una compilación Just in Time (JIT) que

trasforma el código .NET a código nativo dependiendo de la arquitectura sobre la que se esté trabajando.

- Integración de múltiples lenguajes

Sobre la misma plataforma podemos trabajar con lenguajes tales como C#, VB, C++, ASP, J++.

- Gestión de memoria, tratamiento de excepciones, interoperabilidad con código antiguo, sigue el paradigma de programación orientado a objetos.

Arquitectura de .NET Framework

En 2016, Microsoft adquiere Xamarin y lanza la versión .NET Core 1.0, Xamarin fue el responsable en el pasado de que gran parte del Framework .NET se ejecutará bajo el sistema operativo Linux y Unix.

El principal objetivo de construir .NET Core fue la estandarización de diversas plataformas, para que las implementaciones de .NET pudieran compartir las mismas librerías, a esto se le conoce como Librería Estándar de .NET. En la figura 2.10 se muestra la arquitectura de .NET.

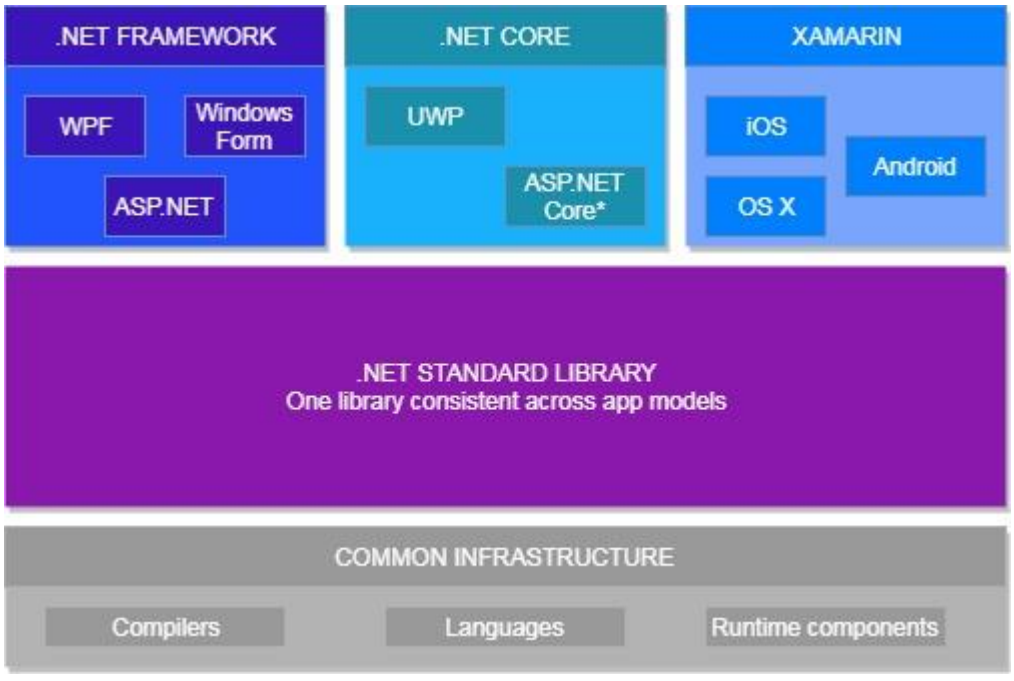


Figura 2.10 Arquitectura .NET

Implementaciones de .NET Framework

- Aplicaciones web basadas en C# desarrolladas a partir de la versión 4.5 del Framework .NET.
- Aplicaciones móviles.
- Videojuegos.
- Aprendizaje de máquinas e inteligencia artificial.

2.2 Breve historia y definición de las bases de datos NoSQL

2.2.1 Definición bases de datos NoSQL

NoSQL hace referencia al término de que generalmente no usa SQL para la manipulación de datos, es comúnmente interpretado como No Solo SQL (Not Only SQL).

Partiendo de esta definición literal, hoy en día, el término va enfocado a los sistemas que no usan los fundamentos de un sistema de gestión de bases de datos relacionales (RDBMS), así como las características de las propiedades ACID, el lenguaje de consulta SQL y los esquemas rígidos de datos, estos aún continúan siendo los sistemas dominantes para aplicaciones empresariales. La razón es porque son eficientes y rápidos para realizar consultas y transacciones, sin embargo son convenientes hasta cierto punto. Si el volumen de datos crece exponencialmente y se requiere obtener los datos con rapidez o si el esquema de datos tiene que modificarse con frecuencia, un sistema NoSQL puede resolver estos requerimientos.

Empresas de impacto mundial están implementando más este tipo de sistemas, una de las principales razones por la cual están migrando a bases de datos NoSQL es porque tienen que manejar grandes volúmenes de información no estructurada en tiempo real, entre ellas podemos encontrar: Facebook, Yahoo, Twitter, Amazon.

2.2.2 Historia de las bases de datos NoSQL

Desde que surgió el término NoSQL en 1998 ha venido evolucionando su definición; en aquella ocasión fue Carlo Strozzi quien utilizó por primera vez este término.

Por medio de Shell Scripting, Carlo realizó consultas a una base de datos en lugar de emplear el ya conocido SQL. Su principal propósito era demostrar mejores alternativas para realizar consultas a una base de datos, sin necesidad de usar las complejas y enormes consultas del tradicional SQL.

Un ejemplo básico del esfuerzo de varios programadores por cambiar la forma de interactuar con la base de datos dejando a un lado parcialmente el uso de SQL, es la librería de Hibernate, la cual basta con realizar el mapeo correcto de las clases en Java con la base de datos, para generar automáticamente sentencias SQL, el programador ya no tiene que preocuparse por escribir las consultas sino que únicamente tiene que llamar a las funciones ya establecidas de la librería de Hibernate.

Todo se hizo con la finalidad de reducir costos, evitar complejidad, mejorar el desarrollo y optimizar el testeo de la aplicación.

En 2006 Google publica una descripción de la estructura de su base de datos distribuida Bigtable. En dicho artículo lo describe como “un sistema de almacenamiento distribuido para administrar datos estructurados que está enfocado a escalar una enorme cantidad datos, rondando los petabytes a través de miles de servidores”.

El funcionamiento básico de Bigtable consiste en almacenar filas con una simple llave principal y los datos se almacenan en filas dentro de familias de columnas relacionadas, para después ser recuperadas tan fácil como si se realizara un join en un sistema de bases de datos relacional.

Bigtable está diseñado para distribuir los datos en múltiples servidores, pero con la ventaja que se accede a los datos de forma más directa en comparación con los sistemas tradicionales de bases de datos.

En el año 2007 Amazon lanza su propio gestor de almacenamiento llamado Dynamo, dicho por la propia empresa “Dynamo es usado para manejar el estado de los servicios que requieren de confiabilidad y necesitan compensar un estricto control entre la disponibilidad, consistencia, costo-eficiencia y rendimiento”

Se describe como un sistema distribuido, basado en un modelo de almacenamiento llave-valor, las llaves son IDs lógicos y el valor puede ser cualquier valor binario.

A partir del año 2009 surgen varios sistemas gestores de bases de datos NOSQL open source tales como MongoDB, Cassandra y se concreta la versión final de Neo4j.

Con el lanzamiento de Bigtable y Amazon muchos proyectos se dieron a conocer gracias a las bases que se establecieron con las primeras versiones del nuevo enfoque NoSQL.

2.2.3 Por qué usar bases de datos NoSQL

Los sistemas NoSQL, se originan a partir de que los sistemas tradicionales de almacenamiento que dominan hoy en día, como los sistemas relacionales, ofrecen una solución poco eficaz y altamente costosa para sistemas de datos distribuidos, para alterar la estructura y el esquema de las tablas en almacenamientos que sobre pasan los cientos o miles de terabytes en unidades de tiempo muy pequeñas y para el manejo de datos semi-estructurados y no estructurados, como, por ejemplo, archivos XML, JSON, PDF, audio, video, imágenes.

Recordemos que las propiedades ACID sobre las cuales se basan los sistemas RDBMS:

- Atomicidad: cuando todas las operaciones que integran a una transacción son exitosas, la transacción se considera exitosa, todas las transacciones son consideradas como exitosas, en caso de una transacción o transacción anidada no se realice con éxito, se deberá de regresar a un estado anterior como si nunca hubieran sucedido dicha transacción.
- Consistencia: Ninguna transacción puede dejar a la base de datos en un estado inconsistente, de esta forma se asegura que la base de datos cambie de un estado valido a otro válido. En caso de pasar a un estado inválido, se deberá regresar al último estado conocido como válido.
- Aislamiento: Ninguna transacción puede afectar a otra transacción concurrente.
- Durabilidad: Las transacciones que fueron completadas satisfactoriamente persistirán en la base de datos, aunque el sistema falle, estos datos permanecerán de alguna forma.

Estas características permiten el correcto funcionamiento de cualquier gestor de bases de datos, sin embargo, ¿Qué pasa con aplicaciones de tipo web-escalar, como las que posee Google, Amazon y Facebook?, donde el almacenamiento de datos crece exponencialmente y manipular toda esa cantidad de información generada no es cosa

sencilla, además requieren una capacidad dinámica de almacenamiento, crecer sin límite, con flexibilidad y agilidad.

Las principales razones para cambiar el enfoque de almacenamiento, es la necesidad de soportar un gran volumen de información en varios servidores, garantizar que la información se encuentre siempre disponible y sobre todo la rapidez para acceder y manipular esa gran cantidad de información en el menor tiempo posible.

De esta necesidad surgen las propiedades BASE, establecido por Eric Brewer, para fundamentar los principios de los sistemas escalables para que sean capaces de soportar este tipo de operaciones.

- Disponibilidad Básica (Basic Availability): Por cada solicitud realizada al sistema se garantiza una respuesta ya sea exitosa o fallida.
- Estado de Software (Soft State): El estado de un sistema puede cambiar dependiendo del paso del tiempo, en ocasiones sin una entrada.
- Consistencia Eventual (Eventual Consistency): Las bases de datos pueden ser momentáneamente inconsistentes, pero se convertirán eventualmente consistentes con el tiempo.

2.2.4 Teorema de Brewer (CAP)

Eric Brewer propone una hipótesis sobre los sistemas distribuidos, en el cual asegura no se puede garantizar que ocurran de forma simultánea las siguientes tres características: consistencia, disponibilidad y tolerancia de partición, como se muestra en la figura 2.11.

Figura 2.11 Teorema de Brewer

- Consistencia: En todos los nodos se ve el mismo dato al mismo tiempo
- Disponibilidad: se garantiza que, por cada solicitud, se devuelve una respuesta ya se exitosa o fallida.
- Tolerancia a Partición: es la capacidad del sistema para continuar ejecutándose, aunque existan divisiones de red o segmentación de la red temporal o permanentemente.

Un sistema distribuido con datos compartidos sólo puede satisfacer dos de estas características al mismo tiempo, por lo que es recomendable utilizar las que más convenga de acuerdo al propósito del sistema.

- CA: Consistencia y Disponibilidad.
- CP: Consistencia y Tolerancia a Particiones.
- AP: Disponibilidad y Tolerancia a Particiones.

Se tiene que elegir entre consistencia o disponibilidad para una red donde se puedan perder mensajes.

Las razones por la cual se debe cambiar el enfoque en el desarrollo de aplicaciones es por las ventajas que ofrecen este tipo de bases de datos.

- Manipular grandes volúmenes de información con una frecuencia alta de lectura y escritura.
- No es necesario definir un esquema de base datos, debido a que la información no esta estructura por lo que puede cambiar en poco tiempo.
- El crecimiento horizontal del hardware es muy versátil, en este tipo de bases de datos es más factible realizar la implementación de un clúster al ir agregando más servidores de acuerdo a la demanda.
- Alta distribución de la información en varios dispositivos o clúster

2.2.5 Tipos de base de datos NoSQL

En el enfoque NoSQL existen 4 tipos de almacenamiento básicos para las bases de datos.

2.2.5.1 Base de datos orientada a documentos

Este tipo de base de datos como su nombre lo indica, almacena y recupera documentos, estos pueden estar en formato de XML, JSON, BSON o YAML. Los documentos almacenados en la base de datos tienen una estructura similar a otros documentos, cada documento puede compartir los mismos atributos con otros documentos, aunque dentro de la misma colección no es necesario que todos los documentos tengan la misma estructura.

La finalidad es evitar la rigidez en la estructura de la información, como en las bases de datos relacionales de Oracle, donde todos los registros poseen la misma cantidad de columnas.

En los siguientes ejemplos se observa cómo se organizan los documentos en formato JSON con diferente estructura.

Ejemplo 1:

```
{
    "ID": "AB123",
    "Nombre" : "Marco",
    "Apellido" : "Rodriguez",
    "Edad" : 45,
    "Salario" : 10000
}
```

Ejemplo 2:

```
{
    "ID": "AB001",
    "Nombre" : "Leticia",
    "Age" : 25,
    "Salary" : 15000
}
```

Ejemplo 3:

```
{
    "ID": "AB9090",
    "Nombre" : "Luis",
    "Edad" : 30,
    "Salario" : 20000,
    "Direccion" : {
         "Numero" : 14,
         "Calle" : "Astros",
         "Estado" : "Nuevo Leon",
         "Pais" : "Mexico"
    }
}
```

El propósito de estas bases de datos es la escalabilidad, si la colección de documentos va creciendo, se puede ir adicionando más servidores al clúster para facilitar la distribución de los datos y optimizar el rendimiento de la aplicación. Haciendo que gracias a estas características estas bases de datos sean muy flexibles para adaptarse a los cambios, por lo que son altamente demandas e implementadas en proyecto que involucran dinamismo en el almacenamiento de datos.

Al realizar una comparativa entre MongoDB y las bases de datos relacionales de Oracle encontramos las siguientes analogías de la tabla 2.12.

MongoDB	Oracle
MongoDB Instancia	DataBase Instancia
DataBase	Esquema
Colección	Tabla
Documento	Fila
_id	Rowid
DBRef	Join

Tabla 2.12 Comparación de objetos MongoDB vs Oracle

Los objetos representados en formato JSON implementan varias reglas para su sintaxis.

- Los datos son organizados en pares de llave-valor
- Comienzan con “{” y terminan con “}”
- Los valores pueden ser numéricos, cadenas de texto, boléanos, arreglos, objetos o con valor NULL.
- Los arreglos se identifican por el uso de corchetes [ ]

Los documentos son accedidos mediante una llave única que representa al documento. Se suele utilizar en aplicaciones Web que trabajan bajo el protocolo HTTP, usando RESTful, al generar índices se puede acceder más rápido a los datos.

No existe un lenguaje estándar para la manipulación de datos, cada base de datos implementa su propio lenguaje para realizar consultas, actualizaciones, inserciones y borrado de datos.

Implementaciones

Las bases de datos orientadas a documentos son recomendadas para datos que no tienen muchas relaciones ni complejas estructuras SQL, son útiles para almacenar logs de los sistemas, preferencias de usuarios, tweets, blogs posts y principalmente para recabar datos de la Web, entre las soluciones más populares podemos encontrar el ya mencionado MongoDB, CouchDB, Terrastone, OrientDB, RavenDB, Lotus Notes,

2.2.5.2 Base de datos orientada a familia de columnas

Almacena y procesa una enorme cantidad de datos distribuidos en varios servidores, los datos son almacenamos en forma de columna en lugar de filas como en las tradicionales bases de datos relacionales, la raíz de la búsqueda es por medio de una llave que está asociada a una super columna, son muy eficientes para realizar lectura, se evita el cruce de información por joins, la desventaja es que no son eficientes para realizar la escritura de datos.

En el siguiente ejemplo de la tabla 2.13, se muestra de forma sencilla como se almacena la información en la base de datos de tipo columna.

ID	Nombre	Apellido	Edad	Salario
0001	Ana	Aguilar	27	15000
0002	Saúl	Rodríguez	31	12000
0003	Carlos	Hernández	35	18000
0004	Elena	García	30	20000

Tabla 2.13 Tabla empleados

Los datos internamente almacenados en sistemas de bases de datos relacionales se realiza de la siguiente forma: Figura 2.14.

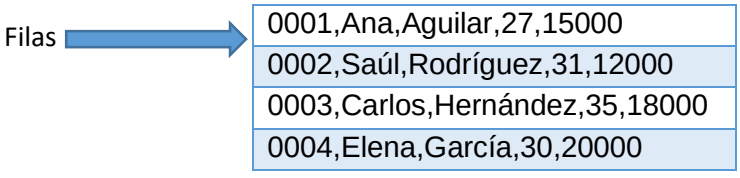


Figura 2.14 Almacenamiento en filas

En cambio, en una base de datos de tipo columna se almacenan los datos como se muestra en la figura 2.15.

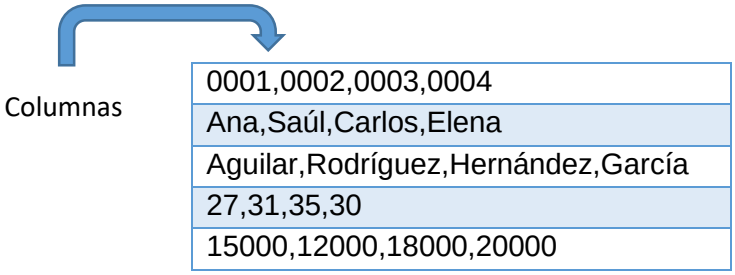


Figura 2.15 Almacenamiento en columnas

Otras características que podemos encontrar en este tipo de base de datos, son que los datos son indexados por un identificador de fila (en inglés Row ID), nombre de columna y un timestamp. Los desarrolladores tienen el control total sobre la ubicación de los datos, no es necesario definir un esquema de datos, agregar columnas nuevas a una familia de columnas, es realmente sencillo.

Implementaciones

Comúnmente se utilizan para Data Warehouses y en sistemas BI, ejemplos comerciales podemos encontrar Apache Cassandra, HBase, Hypertable, Amazon SimpleDB, BigTable, Oracle RDBMS Columnar Expresion.

2.2.5.3 Base de datos orientada a llave-valor

Al igual que otros tipos de bases NoSQL, son altamente distribuidos en un clúster de servidores. Los datos siempre están disponibles.

Es un almacenamiento simple de HashTable, se accede a la base por medio una llave primaria, con la llave se puede obtener, almacenar o borrar un valor, estos valores son de tipo binarios, texto, JSON o XML, generalmente tiene un alto rendimiento y son fácilmente escalables.

La mayoría de las bases de datos de este tipo son eventualmente consistentes, usan el siguiente método para leer los datos:

- En el tiempo que es leído un registro se determina cual es la última llave actualizada valida.

- En caso de que no se pueda determinar la llave más reciente, entonces se muestra todos los valores al usuario para que el mismo decida que valores seleccionar.

A diferencia de un almacenamiento orientado a documentos, donde se crea una llave cuando se genera un nuevo documento, en este tipo de almacenamiento requiere que se especifique la llave, los valores no son indexados y se tiene que conocer forzosamente la llave para poder recuperar el valor.

Estos sistemas de bases de datos distribuidos trabajan en memoria o cache para maximizar el rendimiento. Las consultas se realizan por medio de las llaves, no se puede consultar directamente sobre los valores.

No se recomienda utilizar este tipo de base de datos si lo que se requiere es que existan relaciones entre los datos, tampoco es eficiente para el tema de las transacciones y como ya se mencionó anteriormente no existe forma de realizar una búsqueda por medio de los valores.

Dentro de estas bases de datos, a la agrupación de varios elementos pares llave-valor, se conoce como Bucket, también llamado colección de pares llave-valor. La llave tiene que ser única dentro de cada namespace (Bucket), como se muestra en la figura 2.16.

BUCKET 1	
Key 1	ABC
Key 2	GTF
Key 3	YUR
Key 4	ERF

BUCKET 2	
Key 1	TYH
Key 5	HGB
Key 8	WRD
Key 3	HTF

Tabla 2.16 Colección de pares llave-valor (Bucket)

La rapidez de este tipo de bases de datos se bebe a las llaves, ya que una llave bien definida da una información fundamental para determinar su ubicación dentro de un clúster, así como de sus valores, comúnmente se suele usar particiones en uno o más servidores.

Una llave bien estructurada sirve como identificador único para un valor determinado, responde a consultas sin tener que mirar dentro de los valores.

La manera más eficiente de acceder a las particiones ubicadas en varios servidores, es asignar un rango de llaves a cada uno de ellos, así el administrador sabe perfectamente el rango de llaves que le pertenece a cada servidor.

Para los programadores es más importante tener un eficiente rendimiento al momento de recuperar y almacenar la información, en lugar de estar organizando la información en estructuras complejas como sucede cuando se utilizan tablas.

Implementaciones

Se suele usar este tipo de bases de datos para implementar módulos de logging, en casos donde se requiera alta-velocidad y cacheo de datos, la aplicación más conocida que emplea este tipo de base de datos es Amazon, la cual desarrolló Amazon DynamoDB, que utiliza las ventajas de la alta-velocidad de acceso y técnicas de replicación entre servidores, para la venta de productos on-line, algunas otras bases de datos que podemos encontrar son: Voldemort, Riak, Oracle NoSQL, Redis, Memcached, Berkley DB.

2.2.5.4 Base de datos orientada a grafos

Las bases de datos orientadas a grafos representan una categoría especial dentro de los tipos de bases de datos NoSQL, en donde las relaciones se representan por medio de grafos.

La información está dada por nodos que representan entidades, gráficamente son conocidos como vértices y los enlaces que representan las conexiones entre cada entidad está representado por aristas, debido a su estructura, se puede usar la teoría de grafos para recorrer la base de datos, encontrar la ruta más corta, medidas de centralidad y adyacencias. Un ejemplo de un grafo sencillo de puede observar en la figura 2.17.

Figura 2.17 Representación de un grafo

No se recomienda este tipo de bases de datos para realizar agregaciones, particiones o escalamiento horizontal, ya que recorrer grafos localizados en distintas

ubicaciones puedes ser costoso y difícil de acceder; si lo que se requiere es mejorar el rendimiento de la aplicación, se puede realizar escalamiento vertical.

El propósito de estas bases de datos es ver a las cosas con las que interactuamos todos los días como una red, donde todo relativamente tiene una conexión o relación con otros objetos, partiendo de este enfoque, podemos empezar a resolver problemas complejos, construir redes como por ejemplo, en las composiciones químicas de las moléculas, la estructura de una proteína, en las redes sociales, en el sistema de transporte de trenes, en las autopistas y carretas que unen a las ciudades, etc., realmente casi todo lo que nos rodea lo podemos ver como una red de conexiones.

En este tipo de base de datos, tiene la capacidad de consultar información para descubrir patrones en los datos y para realizar análisis profundo, es muy similar a la minería de datos aplicada al análisis de datos, solo que para este enfoque se utilizan algoritmos de minería de grafos, uno de los más conocidos es el sistema Subdue, que es útil para descubrir patrones en los grafos, además también se puede implementar modelos matemáticos.

Un grafo simple puede ser representado como una matriz, la cual es llamada matriz de adyacencia. La matriz es cuadrada n x n y depende del número de vértices del grafo, por ejemplo en la figura 2.18 se muestra este tipo de grafo:

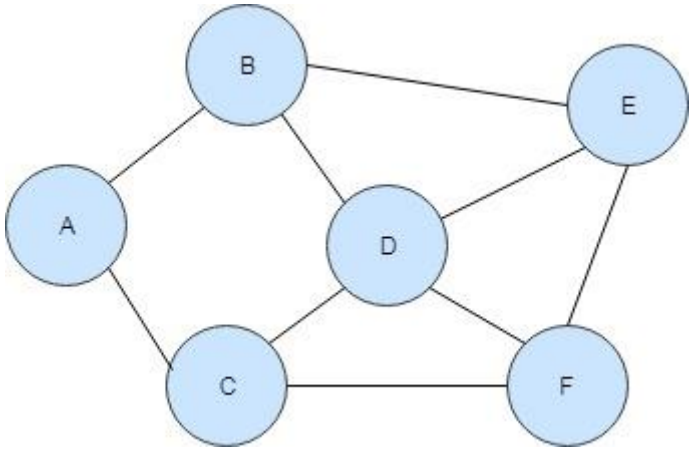


Figura 2.18 Grafo de relaciones

En este caso como el grafo tiene 6 vértices entonces la matriz será de 6 filas x 6 columnas, cada fila y columna representa un nodo del grafo, como se muestra en la figura 2.19.

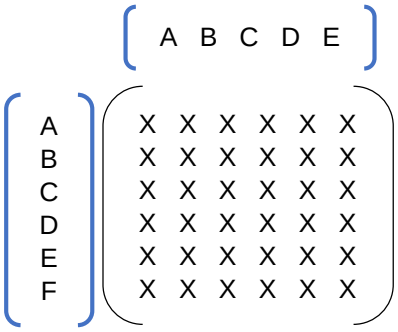


Figura 2.19 Matriz n x n

Una de las formas para realizar el llenado de la matriz, aunque existen varios métodos y teoremas de cómo hacer esta traducción, es colocar 1 o 0 donde exista una relación entre cada nodo, como se muestra en la figura 2.20.

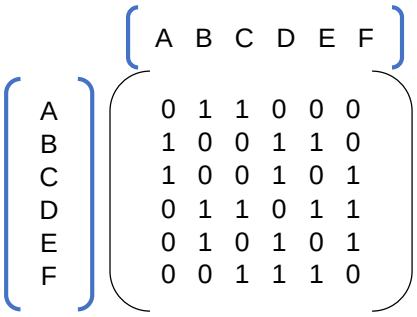


Figura 2.20 Representación matricial de un grafo

Al convertir el grafo a una matriz de $n \times n$ podemos aplicar operaciones matriciales y obtener ecuaciones para aplicar modelos matemáticos.

Implementaciones

Por lo general estas bases de datos se suele usar en aplicaciones muy complejas sobre todo para el análisis de las estructuras que conforman al grafo, como en minería de estructuras bioquímicas, análisis de genomas, programas para el análisis del control de

flujos, minería en redes de comunicación, análisis de redes sociales, en páginas web donde se muestra sugerencias al usuario, manejo de datos maestros y aplicaciones para la detección de fraudes, entre otros. Algunas bases de datos conocidas son: Neo4J, FlockDB, GraphBase, InfoGrip, OrientDB.

2.2.6 Persistencia poliglota

Desde que aparecieron las bases de datos relacionales, se popularizó demasiado esta nueva forma de almacenar información, tanto que muchos programadores comenzaron a ver la forma de cómo integrar este tipo de tecnología a sus aplicativos, en aquel entonces funcionaban bajo mainframes, y desde entonces, la metodología para almacenar y manipular grandes volúmenes de información ha evolucionado de tal forma que sea más eficiente y menos costoso adaptarse a las nuevas necesidades del negocio.

En la actualidad muchas instituciones financieras continúan utilizando los sistemas mainframe para su operación diaria. Una de las principales ventajas que ofrecen las bases de datos relaciones es la consistencia de los datos, que hace que las transacciones sean confiables, por lo que las bases de datos relacionales todavía se seguirán utilizando, en algunos casos la mejor solución es implementar este tipo de bases de datos.

En el año 2006 Nel Ford, introduce el término programación poliglota para exponer una alternativa de desarrollar una aplicación con múltiples lenguajes de programación, aprovechando las ventajas de cada uno de ellos para resolver problemas específicos.

Explica que es más productivo utilizar el lenguaje correcto para dar solución a cada uno de los problemas que se presentan en los componentes de la aplicación, que intentar resolver todo con un solo lenguaje.

La persistencia poliglota hace referencia a la programación poliglota y se define como la capacidad de usar diferentes tipos de bases de datos en una aplicación para dar solución a diferentes problemas, es decir, de acuerdo a las características y ventajas que ofrece cada sistema manejador de bases de datos, se tiene que analizar e implementar una solución adecuada para cada módulo que conforma la aplicación.

Como ya se mencionó anteriormente una de las principales características que fortalecen a las bases de datos relacionales es su transaccionalidad, de igual forma las bases de datos NoSQL también poseen cualidades que ayudan a mejorar el rendimiento del sistema.

En caso de que se requiera manejar relaciones en los datos entonces lo que se necesita es un tipo de almacenamiento de grafo. Si el almacenamiento de tipo columna no puede manejar una estructura muy compleja entonces es mejor utilizar el almacenamiento de tipo documentos. El almacenamiento de tipo columna es conveniente utilizarlo para operaciones de agregación ya sea para realizar un conteo, promedio o suma.

Muchas empresas suelen tener un solo motor de base de datos para almacenar los datos de su negocio, como los datos de logging, transacciones del negocio, datos de reporteo, BI y data warehousing, como se muestra en la figura 2.21.

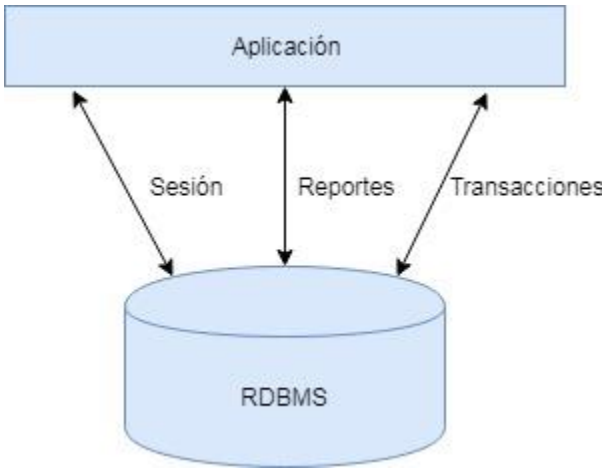


Figura 2.21 Aplicación con un solo sistema de base de datos

Considerando las características de cada sistema de base de datos y aprovechando sus ventajas para optimizar el acceso, las transacciones y el almacenamiento de los datos podemos reestructurar la arquitectura como se muestra en la figura 2.22.

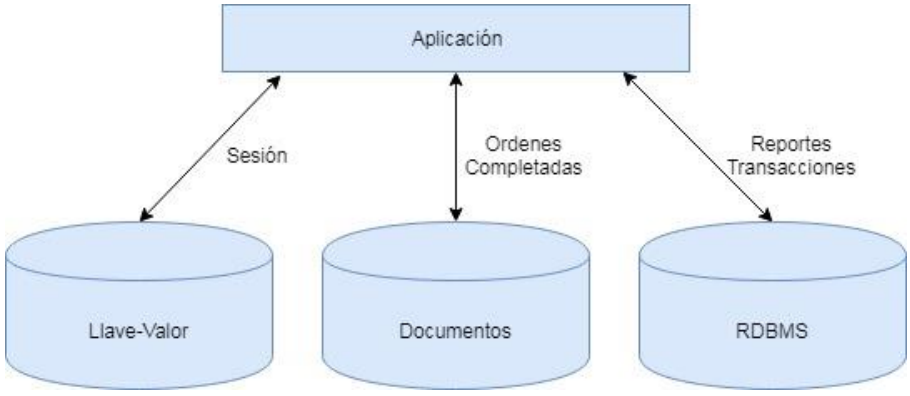


Figura 2.22 Aplicación con varios sistemas de bases de datos

No necesariamente se tiene que usar una simple base de datos para resolver todos los problemas, dependiendo del propósito de la aplicación se puede utilizar varias bases de datos y lograr su integración por medio de ETLs.

2.2.7 Modelos de datos NoSQL

En esta sección se abordan temas donde se muestra la estructura lógica de las bases de datos, manejando diferentes enfoques como las relaciones, almacenamiento, acceso y manipulación de la información.

2.2.7.1 Modelo de datos de agregación

Una agregación es un una colección de datos agrupados como si fueran una sola unidad, hace que sea más fácil manejar los datos almacenados sobre clúster. Este modelo va enfocado principalmente a bases de datos de tipo llave-valor, documentos y a familia de columnas.

Las bases de datos llave-valor y documentos están construidas básicamente sobre este enfoque de agregación, cada agregación que se forma tiene una llave principal o ID que sirve para extraer la información, por ejemplo en las bases de datos de tipo llave-valor, la agregación y la búsqueda se realiza por medio de su llave, en cambio para las bases de datos orientadas a documentos, la agregación y la búsqueda se realiza por medio de una etiqueta que puede ser indexada para agilizar la recuperación de datos.

Las bases de datos orientadas a columnas, la agregación se tiene que ver como una estructura de dos niveles, es decir, el primer nivel consiste en obtener la llave identificadora de la agregación y en segundo nivel obtener los identificadores de las columnas para posteriormente obtener los datos, como se ejemplifica en la figura 2.23.

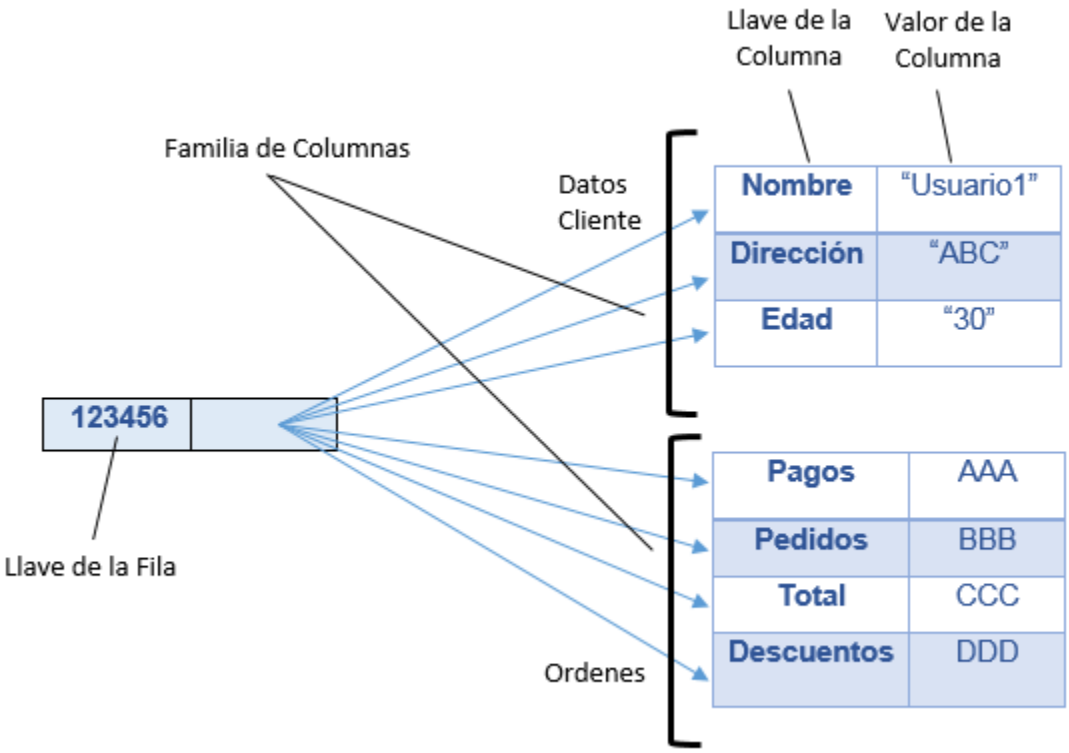


Figura 2.23 Agregación por columnas

Las bases de datos orientados a columnas agrupan la información dentro de una familia de columnas, la cual representa un registro, de esta forma podemos ver al conjunto de una familia de columnas como un join, donde tenemos la información agrupada, tienen en relación una llave principal para poder hacer la recuperación de los datos.

2.2.7.2 Modelo de datos distribuido

La necesidad de ejecutar bases de datos en grandes clusters y manipular información en constante crecimiento, hacen que escalar de forma horizontal este tipo de sistemas se vuelva muy complicado y costoso.

Los modelos distribuidos dan parte de la solución para este tipo de problemas y dependiendo del modelo utilizado, será la habilidad para manejar grandes volúmenes de datos en lectura y escritura. Generalmente existen dos maneras de realizar la distribución de los datos, replicación y sharding, aunque también existe una forma muy simple y sencilla la cual es colocar todos los datos en un solo servidor, a esto se le llama "*single server*".

La replicación consiste básicamente en colocar los mismos datos en varios servidores y se subdivide en dos subcategorías, replicación master-slave y peer-to-peer, en el caso de sharding consiste en colocar diferentes datos en diferentes servidores.

Single Server

Este tipo de distribución es la más simple y la más recomendable, aquí se centralizan todos los datos en un solo servidor comúnmente, se utiliza para realizar una gran cantidad de operaciones de agregación a la base de datos, se suele ocupar bases de datos como las orientadas a documentos o de llave-valor, así los programadores eliminan la complejidad de configurar el cluster de la base de datos.

Sharding

Este tipo de almacenamiento distribuido coloca datos diferentes en nodos separados, con la finalidad de hacer un balanceo de cargas para evitar saturar un nodo en específico, así cada nodo se encarga de una cierta cantidad de usuarios para obtener una respuesta eficiente. Relativamente es convertir una gran base de datos en varias bases de datos pequeñas, como se muestra en la figura 2.24.

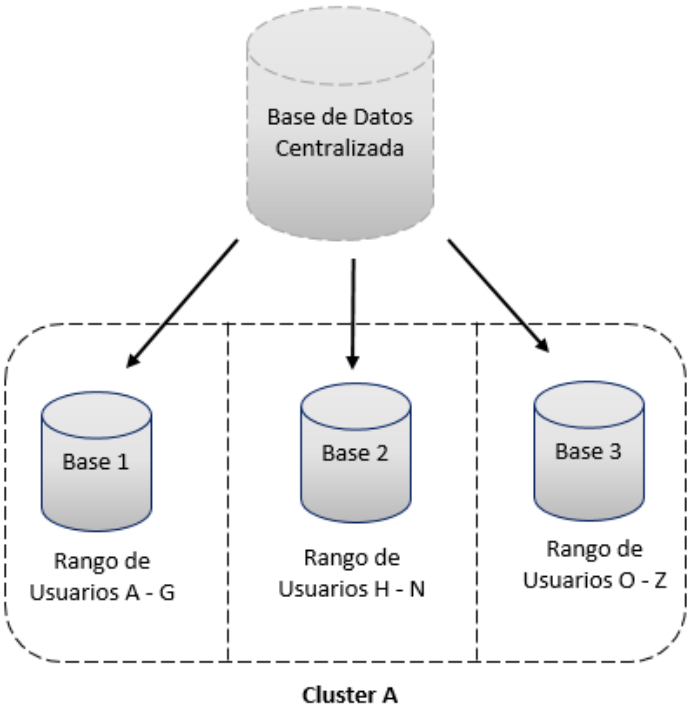


Figura 2.24 Sharding en Bases de Datos

Este tipo de información se debe de agrupar de acuerdo a un patrón de acceso, esto quiere decir que se debe de agrupar todos los datos que tienen relación entre sí, para asegurar que los datos comúnmente accedidos, se ubiquen dentro del mismo servidor.

Replicación Master-Slave

Con este tipo de distribución, la información se replica por varios servidores, a través de un nodo maestro o primario, el cual es la fuente de información para todos los servidores o nodos esclavos también conocidos como secundarios, en general un proceso de replicación consiste en sincronizar todos los nodos secundarios con base al nodo principal. Para garantizar la consistencia de los datos, todas las operaciones de actualización se realizan en el nodo principal, a partir de aquí se propaga la información a los nodos secundarios, así como se muestra en la figura 2.25.

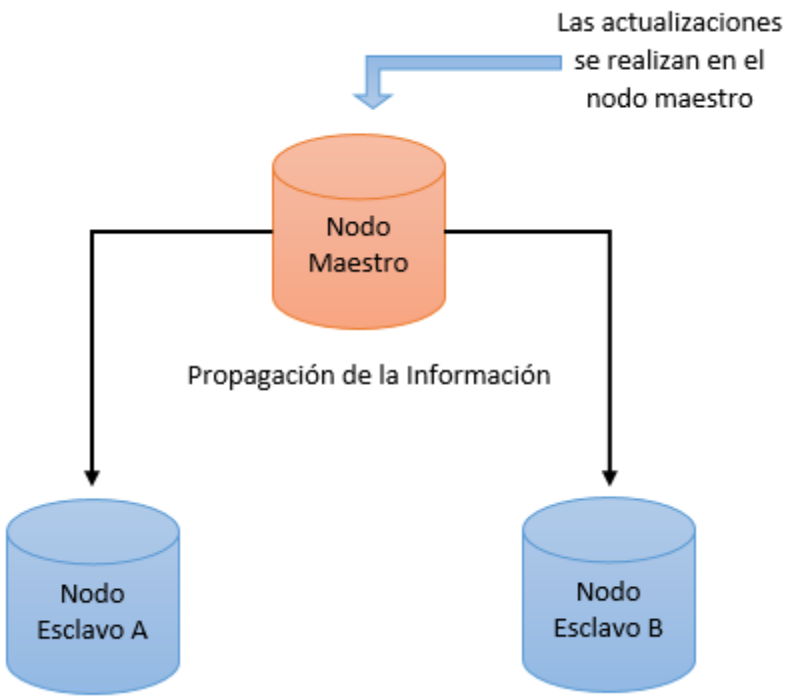


Figura 2.25 Replicación desde el nodo maestro

Todas las escrituras se realizan en el nodo maestro y las lecturas se pueden realizar desde el nodo maestro o los nodos esclavos. Este tipo de distribución es ideal para las lecturas intensivas que se realizan sobre la base de datos, sin embargo no es la arquitectura adecuada para aplicaciones que requieran escrituras intensivas, debido a que puede llegar a saturar al nodo maestro.

Otro beneficio de este tipo de distribución es la lectura resiliente, en caso de que falle el nodo principal, se puede seguir atendiendo las solicitudes desde los nodos esclavos, si falla el nodo principal, la desventaja es que mientras no se reestablezca, no se podrán realizar actualizaciones en la base de datos. Para colocar un nuevo servidor maestro, resulta de cierta forma fácil, se puede restaurar toda la información desde un nodo esclavo.

Replicación Per-To-Per

El inconveniente de utilizar la metodología de Master-Slave, es que existe un cuello de botella en el nodo principal, para suprimir este problema, se formula un nuevo método, el cual consiste en replicar toda la información en todos los nodos y habilitar la lectura y escritura en todos los nodos disponibles del sistema. Un ejemplo de ello se puede observar en la figura 2.26.

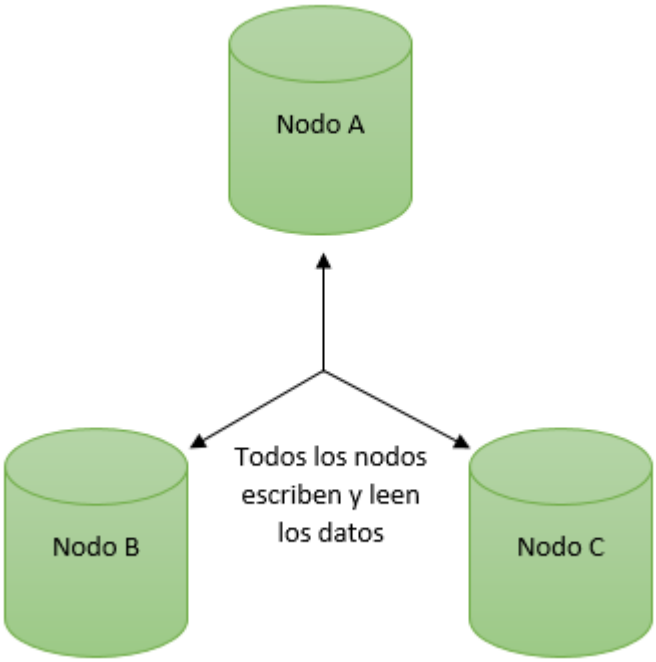


Figura 2.26 Replicación Per to Per

En caso de que falle un nodo, no tiene gran relevancia, ya que los demás nodos continúan trabajando, el inconveniente de este tipo de distribución es la consistencia. Cuando dos usuarios estén intentando escribir sobre el mismo registró al mismo tiempo, generaría un serio problema en la consistencia de los datos. Para atender este tipo de temas se tiene que implementar un mecanismo para coordinar y sincronizar las escrituras en los nodos.

2.2.7.3 Consistencia

El término de consistencia en una base de datos distribuida, se refiere fundamentalmente a que todos los datos que son replicados en varios nodos, posean los mismos valores.

Al intentar replicar, conlleva varios problemas, uno de ellos y el más popular es el conflicto de escritura-escritura, es cuando dos personas intentan escribir sobre el mismo registro al mismo tiempo, ocasionando inconsistencia en los datos. Para solucionar estos conflictos, se han creado varios algoritmos de control de concurrencias, algoritmos de marcas de tiempo y hasta mostrar en pantalla los datos actualizados para que el propio usuario decida que dato deberá permanecer.

Para el caso de las bases de datos NoSQL, se suele utilizar la terminología consistencia eventual, generalmente las bases de datos NoSQL están enfocadas a sistemas distribuidos, al realizarse la réplica de los datos en los diferentes nodos del cluster, no se garantiza de que en ese instante todos los datos sean consistentes.

Puede ocurrir que en algún momento de la actualización se realice una consulta de datos, en ese preciso momento los datos aún no se encuentran con la última versión del valor actualizado, ocasionando que los datos leídos sean obsoletos o se realicen lecturas sucias.

En esencia los sistemas NoSQL implementan los principios de BASE (*Basically Available Soft-state Eventual Consistency*) mencionados anteriormente, que en conjunto con los conceptos de teorema de CAP, se tiene que hacer un balance entre disponibilidad, consistencia y tolerancia a particiones, dar prioridad a unas características y dejar de lado otras características no tan relevantes, ya que solo se puede usar un par de estas características a la vez, dependiendo del propósito del sistema y las condiciones de operación.

2.2.7.4 Map-Reduce

Es un patrón que permite hacer operaciones computacionales sobre un cluster, reduce las tareas tomando muchos valores y genera una salida simple con los valores sumariados utilizando el modelo de datos de agregación, tiene la ventaja de que se puede paralelizar las operaciones. Su salida está en formato de llave-valor, si se requiere de un

uso constante de una operación map-reduce, es recomendable almacenarlo como vista materializada.

Se generan valores llave-valor con los datos seleccionados por medio de la operación Map, como se muestra en la figura 2.27.

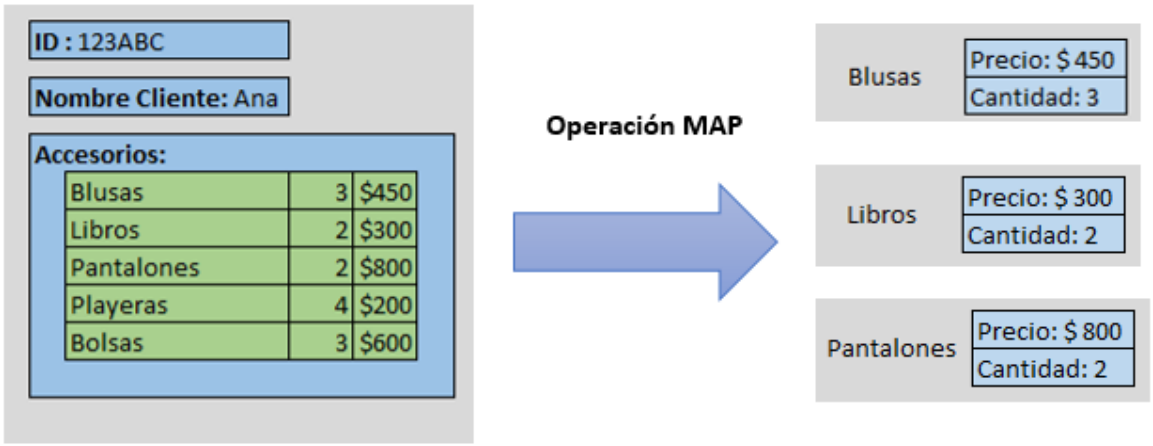


Figura 2.27 Operación Map, generación de pares llave-valor

En la figura 2.28, se realiza la agregación de los datos seleccionados con la operación Reduce.

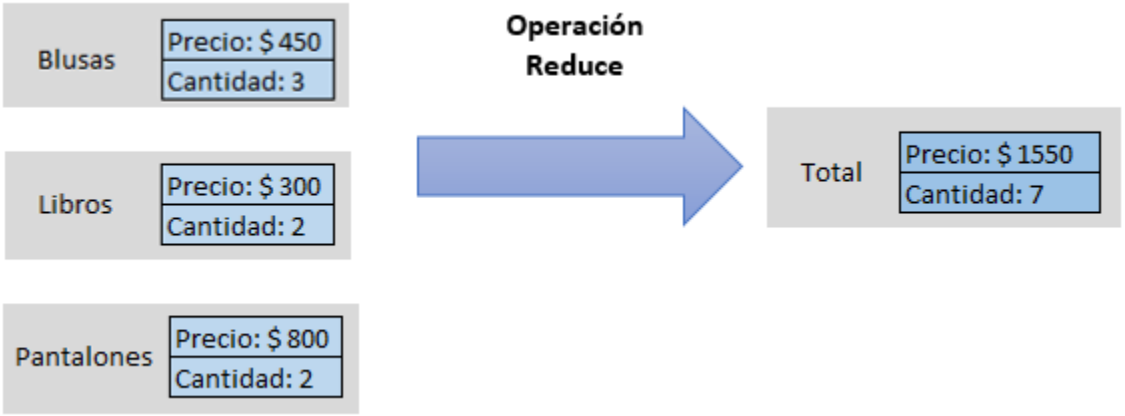


Figura 2.28 Operación Reduce, agregación

El framework map-reduce, son dos operaciones que trabajan conjuntamente, primero se leen varios registros de la base de datos para generar pares de llave-valor y en segundo lugar con la función Reduce toma varios pares de llave-valor y agrupa los valores dentro de una sola llave, para simplificar el resultado. .

2.2.8 Ejemplos bases de datos NoSQL

En la actualidad existe un sinfín de bases de datos NoSQL, cada una de ellas se enfoca a solucionar distintos problemas, algunas bases de datos fueron desarrolladas por empresas especializas en TI que brindan servicios en la nube, para cubrir sus propias necesidades, por ejemplo, escalabilidad, rapidez para consultar datos, crecimiento exponencial de datos, análisis, descubrimientos de patrones, etc. A continuación se realiza un breve análisis sobre algunas de estas bases de datos.

2.2.8.1 MongoDB

Descripción

MongoDB está enfocado principalmente para el desarrollo web y está construido para trabajar bajo los protocolos y la infraestructura de internet, además la escalabilidad hace que el crecimiento de los datos no sea un problema al momento de agregar nuevos servidores.

Su modelo de datos optimiza su rendimiento para agilizar la lectura y escritura de datos, principalmente la razón por la cual se elige MongoDB es porque es una base de datos orientada a documentos, en lugar de estar estructurada por filas y columnas, esto es una gran ventaja, ya que no sigue estructuras de datos rígidas, posee un enorme potencial para trabajar con datos no estructurados.

Un ejemplo de documento en formato JSON de MongoDB, se observa en la figura 2.29.

```
{
    _id: ABC1234,
    nombre: 'Lety',
    clave:’1234’,
    email: ‘lety@gmail.com'
}
```

Figura 2.29 Documento JSON

También pueden existir documentos anidados o atributos compuestos en arreglos, como se muestra en la figura 2.30.

```
{
    _id: ABC1234,
    nombre: 'Leticia',
    clave:’1234’,
    email: [‘lety@gmail.com',
          let123@hotmail.com],
       direccion:{
          calle:’Nacional’,
          colonia:’Valle de Bravo’,
          No:’18’,
          Estado:’Mexico’
       }
}
```

Figura 2.30 Documento JSON Anidado

Como se observa, este tipo de base de datos ofrece una enorme agilidad para almacenar información variable y de todo tipo, para después ser consultada por medio de una llave.

Modelo de datos

MongoDB usa el modelo de datos orientado a documentos, internamente almacena los documentos en formato llamado Binary JSON o BSON los cuales están destinados a almacenar múltiples documentos. Cuando se ejecuta una consulta a la base de MongoDB, la respuesta igualmente está en formato JSON.

MongoDB utiliza colecciones en lugar de tablas si lo comparamos con una tradicional base de datos relacional y en lugar de tener registros existen documentos, esto quiere decir que la información se almacena en colecciones de documentos. Los datos en una colección son almacenados en disco y para consultar algún dato es necesario indicar a que colección pertenece.

La idea de esquema en MongoDB es dinámica, lo que significa que cada documento puede tener diferentes tipos de campos, como son comentarios, arreglos, documentos embebidos, texto, datos de tipo numérico, fechas, etc. Gracias a su modelo de datos, ofrece flexibilidad para datos polimórficos al no respetar una estructura de datos.

MongoDB ofrece una arquitectura de almacenamiento flexible, optimizado para aplicaciones de alta demanda, principalmente para lectura y escritura en servidores distribuidos. En la figura 2.31 se muestra la arquitectura básica.

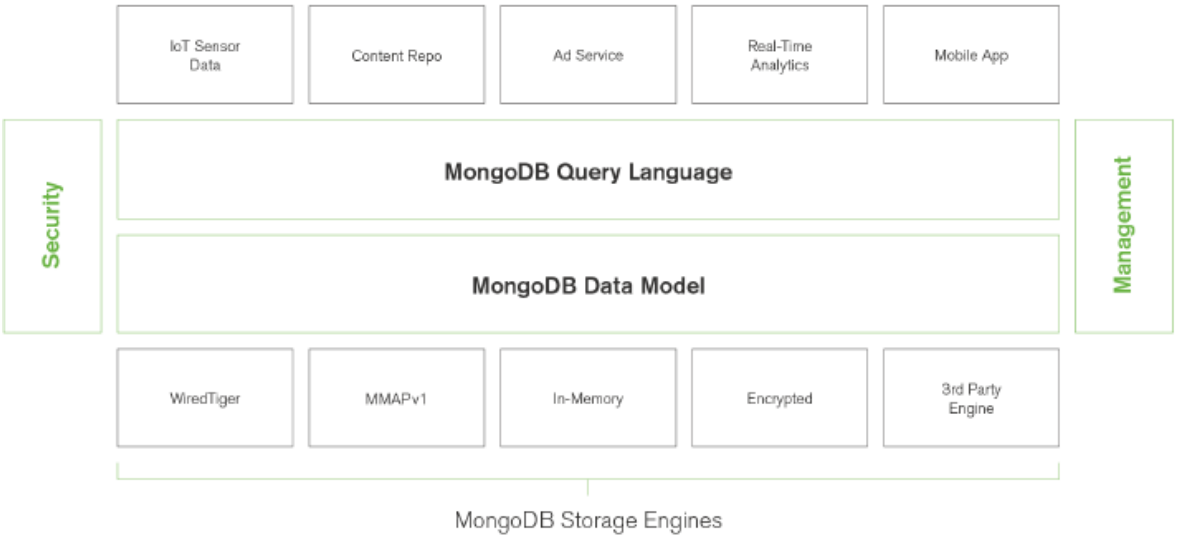


Figura 2.31 Arquitectura MongoDB

Tipo de consultas

En MongoDB existen operaciones básicas sobre la base de datos, similares a las operaciones de las bases de datos relacionales, algunas de estas operaciones que podemos encontrar son insertar, actualizar, borrar, etc.

Suponiendo que tenemos una colección llamada *empleados* en la base de datos MongoDB y en su contraparte, en la base de datos relacional una tabla llamada igualmente *empleados,* entonces podemos ejecutar las siguientes operaciones, de la tabla 2.32:

Acción	Comando MongoDB	Comando SQL
Seleccionar todos los empleados	db.empleados.find()	select * from empleados
Insertar empleados	db.empleados.insert({ campo: valor })	insert into empleados(campos) values (valor)
Actualizar un empleado	db.empleados.update({ condición } , { campo: valor })	update empleados set campo = valor where condición
Contar el número de empleados	db.empleados.count():	select count(*) from empleados

Acción	Comando MongoDB	Comando SQL
Borrar un empleado	db.empleados.remove({condición}):	delete from empleados where condición
Eliminar todos los empleados	db.empleados.drop().	truncate table empleados

Tabla 2.32 Comparativa de comandos MongoDB vs SQL

Map-Reduce

Dentro de MongoDB, existe la función mapReduce la cual consiste básicamente como ya se describió anteriormente, en tomar un gran volumen de datos, obtener valores llave-valor, aplicar una función de agregación y finalmente extraer su resultado, como se muestra en la figura 2.33.

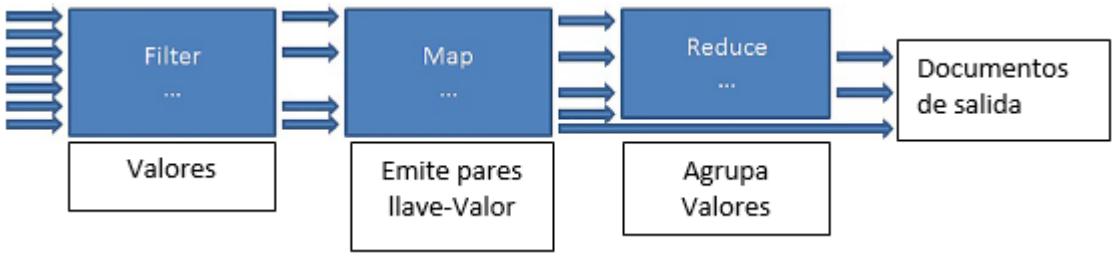


Figura 2.33 Diagrama Map-Reduce en MongoDB

En el ejemplo de la figura 2.34 se puede observar cómo se implementa la función `mapReduce`, a partir de las condiciones que se indican en la consulta (`query`), se obtienen pares de llave - valor (`key – values`) con base a los campos `cust_id` y `amount`, una vez que se obtienen estos valores se les aplica la función `sum`, que es una función de agregación que suma los valores obtenidos de cada llave, finalmente se guarda el resultado en una colección llamada `order_totals`, el ejemplo fue tomado de la página de documentación de MongoDB[7].

[7] https://docs.mongodb.com/manual/core/map-reduce/

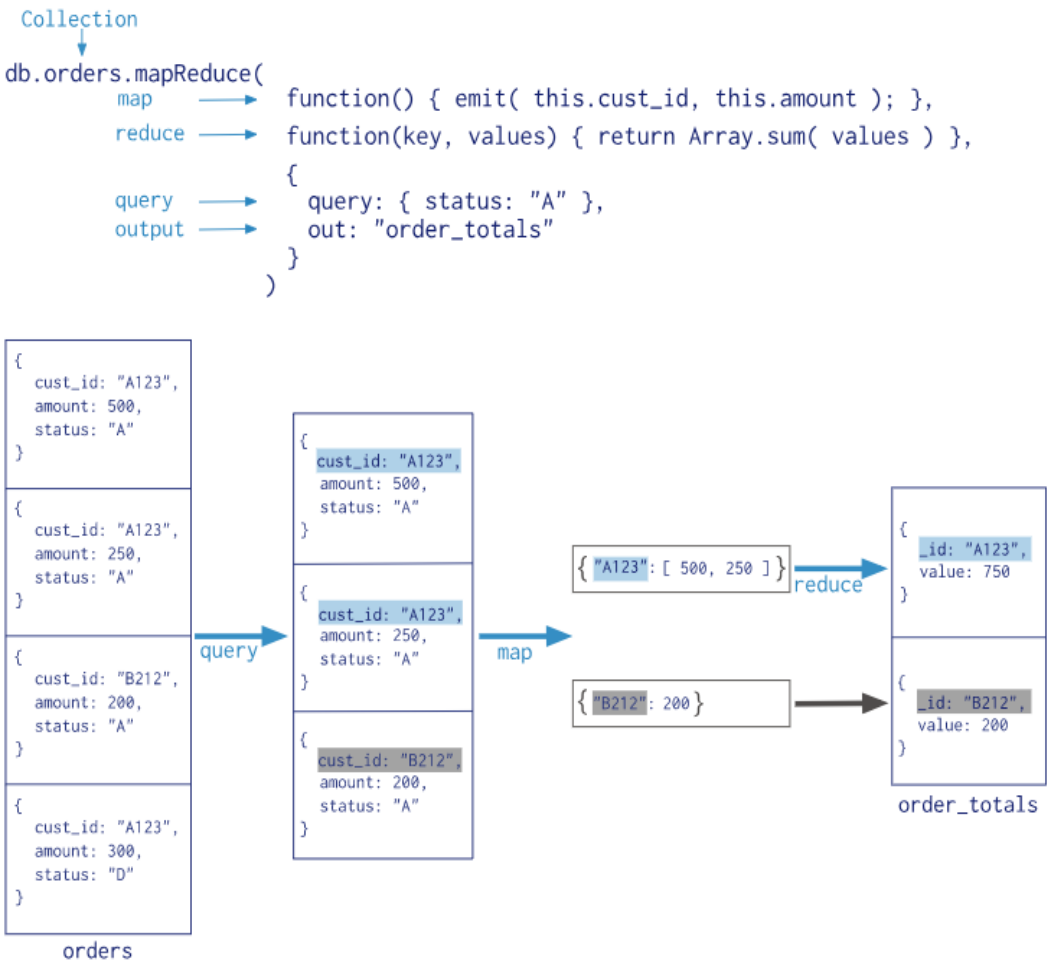


Figura 2.34 Ejemplo Map-Reduce en MongoDB

Índices

Los índices en la base de datos funcionan como los índices en los libros, agiliza la búsqueda de un contenido específico por medio del número de página, en una base de datos se brinda la misma funcionalidad.

En MongoDB los índices son implementados como una estructura de datos llamada B-tree, estas estructuras son optimizadas para una variedad de consultas, incluyendo escáner de rangos y consultas con cláusulas de ordenamiento. El índice por lo general se asigna automáticamente a la llave primaria que sirve como identificador único para acceder eficientemente a los datos. También existen los llamados índices secundarios, que sirven para mejorar la rapidez de las consultas sobre los demás campos.

En MongoDB por colección se puede crear hasta 64 índices incluyendo a los de tipo unique, hashed, llave compuesta, ascendiente, descendiente, texto e incluso índices geoespaciales.

Para generar índices sobre las colecciones se utiliza el siguiente comando:

```
db.nombreColeccion.createIndex({campo},{opciones})
```

Arquitectura de replica

El tipo de replica que ofrece MongoDB se conoce como Conjuntos de Replica (*Replica Sets)*, consiste en distribuir información a través de 2 o más servidores, la replicación es usada para optimizar el acceso de lectura a las bases de datos, en caso de tener una aplicación que este solicitando constantemente datos que comúnmente ocurre en aplicaciones web.

La replicación es de tipo Master-Slave donde solo existe un solo servidor central para realizar todas las actualizaciones o escrituras a la base, además existe uno o varios servidores que hacen la función de esclavos, cuya función únicamente es ser accedidos solo para lectura de datos.

En caso de que falle el servidor maestro, en automático se sustituye por un servidor esclavo y comienza a realizar todas las funciones del servidor central. En cuanto se reestablece el servidor central, este ya no vuelve a ser el servidor maestro sino que se queda como servidor esclavo, a esto se le conoce como redundancia y conmutación automatizada de errores. En la figura 2.35 se muestra este tipo de redundancia tolerante a fallos.

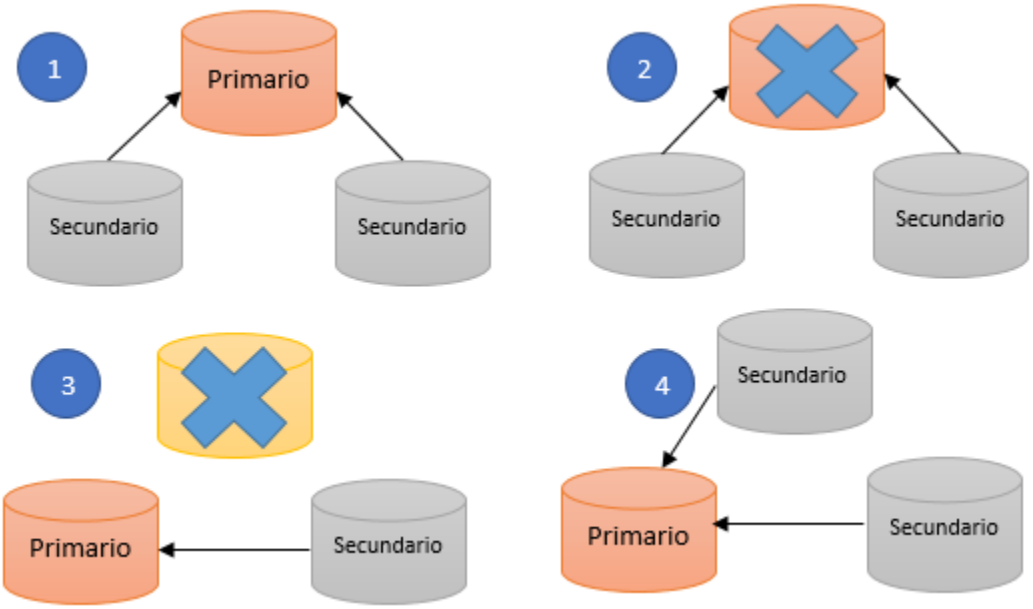


Figura 2.35 Replicación en MongoDB

La replicación es una de las características más usadas en este tipo de base de datos debido a su eficiencia y alta tolerancia a fallas.

2.2.8.2 Neo4J

Descripción

La base de datos Neo4j fue desarrollada por la empresa Neo Technology Inc. Su primera versión fue lanzada en el año 2007, funciona bajo la licencia GNU y AGPL.

Neo4j es una base de datos orientada a grafos, representa la relación entre dos o más objetos, por medio de elementos llamados vértices (nodo) y aristas (relación), aunque la representación más sencilla, es un simple nodo sin relaciones.

Modelo de datos

Los elementos fundamentales en el modelo orientado a grafos es el nodo que suele representar entidades y las relaciones que representan una asociación entre 2 o más nodos, los cuales pueden contener propiedades. Neo4j se basa en los principios de la teoría de grafos y conceptos matemáticos para obtener un poderoso y eficiente motor de búsqueda de datos.

Al conjunto de nodos, relaciones y propiedades son el equivalente a las columnas en el modelo relacional. Para relacionar los nodos, estas relaciones son unidireccionales o bidireccionales.

En el grafo de la figura 2.36, se observa cómo se puede clasificar a los nodos en varios tipos:

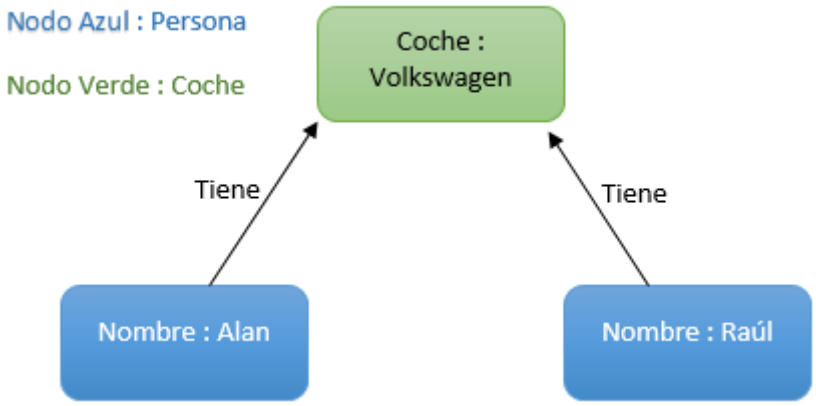


Figura 2.36 Clasificación de nodos en Neo4j

Para realizar el modelado de convertirlo los datos en grafo, es indispensable identificar un dominio o alcance de datos, posteriormente definir los nodos y las relaciones, finalmente convertir el modelo a lenguaje de Cypher, que es propio de Neo4j.

Generalmente para realizar esta traducción los nombres de objetos, personas, lugares junto con sus propiedades se convierten en nodo y las acciones o verbos se representan como una relación, también las relaciones pueden tener propiedades para indicar la magnitud de la relación.

Un *traversal* es la forma de cómo se consulta un grafo, se realiza un recorrido por varias rutas para obtener un resultado, se hacen visitas a cada nodo de acuerdo a las reglas establecidas, siguiendo una relación en los patrones encontrados. El nodo inicial para comenzar a realizar la búsqueda se llama *Starting Node*.

Tipo de consultas

Cypher es el lenguaje de consulta declarativo de Neo4j, quien se encarga de buscar patrones en los nodos y en las relaciones del grafo para extraer la información. Tiene la capacidad de crear, actualizar, remover nodos, relaciones, etiquetas y propiedades, a su

vez también maneja índices y constraints. Su enfoque para recuperar los datos se basa en qué se quiere consultar, más no, en cómo se hace esa recuperación de datos.

La estructura de Cypher es similar a la de SQL, de donde se toman varias cláusulas, algunas de ellas son: CREATE, DELETE, ORDER BY, WHERE, SET, MERGE.

La sintaxis de `Cypher` se muestra en la figura 2.37.

```
[MATCH WHERE]
[OPTIONAL MATCH WHERE]
[WITH [ORDER BY] [SKIP] [LIMIT]]
RETURN [ORDER BY] [SKIP] [LIMIT]
```

Figura 2.37 Sintaxis Cypher

En el ejemplo de la figura 2.38, se ilustra una búsqueda dentro de un grafo. Considerar el siguiente grafo:

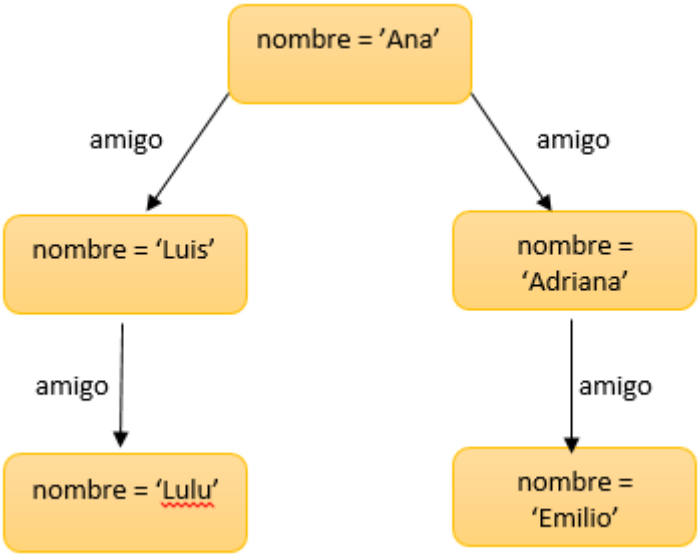


Figura 2.38 Grafo de relaciones

En el ejemplo de la figura 2.39, se desea obtener los amigos de los amigos de Ana:

```
MATCH (ana {nombre: 'Ana'})-[:amigo]->()-[:amigo]->(ada)
RETURN ana.nombre, ada.nombre
```

Figura 2.39 Consulta en Cypher

Obteniendo el siguiente resultado de la figura 2.40.

```
+-------------------------+
| ana.nombre | ada.nombre |
+-------------------------+
| "Ana"      |   "Lulu"    |
| "Ana"      |   "Emilio"  |
+-------------------------+
2 rows
```

Figura 2.40 Resultado de la consulta

Neo4j contiene librerías que implementan poderosos algoritmos de búsquedas, cada uno de ellos se implementa de acuerdo al propósito del descubrimiento de las relaciones en los grafos:

- PageRank
- Betweenness Centrality
- Closeness Centrality
- Louvain
- Label Propagation
- Connected Components - Weakly
- Connected Components - Strongly
- Clustering Coefficient - Triangle Count
- Minimum Weight Spanning Tree
- All Pairs- and Single Source – Shortest Path
- A* Algorithm
- Yen’s K-Shortest Paths

Map-reduce

Neo4j no implementa la función map-reduce, sin embargo, se apoya de la herramienta Apache Hadoop, en conjunto ofrecen características muy sofisticadas para el análisis de datos.

Apache Hadoop es un framework escrito en Java y maneja su propio sistema de archivos llamada HDFS (Hadoop Distributed File System), está enfocado al desarrollo del Big Data.

Su función principal es tomar un archivo de gran tamaño y separarlo en varios archivos pequeños, para distribuirlo en los diferentes nodos que conformar el cluster, esto con la finalidad de trabajar por separado y de forma paralela el procesamiento de cada archivo, cuando termina el procesamiento de los archivos, el mismo proceso se encarga de unir todos los resultados para mostrar un solo resultado.

Índices

Para optimizar las búsquedas en los nodos y aumentar su velocidad de procesamiento, se implementan índices sobre las propiedades de todos los nodos que poseen alguna etiqueta o clasificación en común, para mejorar el tiempo de respuesta de las búsquedas.

Los índices en Neo4j son eventualmente disponibles, esto quiere decir que cuando se crean los índices, el proceso de creación regresa una respuesta inmediatamente, pero esto no quiere decir que ya esté disponible el índice inmediatamente para realizar consultas, la creación del índice se lanza a *background* y hasta que esté completamente finalizada la creación del índice, entonces se podrá obtener una respuesta más rápida en las consultas.

Se tiene que tener cuidado al momento de indexar los datos, ya que un índice implica espacio en disco y escritura lenta, por lo que es sugerible indexar aquellos datos que sean relevantes o claves para realizar las búsquedas.

Para crear un índice sobre nodos que poseen una etiqueta en común, se tiene que especificar también la propiedad sobre la que se va a realizar la indexación.

Se utiliza el siguiente comando:

```
CREATE INDEX ON :Etiqueta(Propiedad)
```

Ejemplo:

```
CREATE INDEX ON :Persona(nombre)
```

Para crear índices compuesto sobre varias propiedades.

```
CREATE INDEX ON :Etiqueta(Propiedad1, …, PropiedadN)
```

Ejemplo:

```
CREATE INDEX ON : Persona (nombre,edad,fechaNacimiento)
```

Arquitectura de replica

En este capítulo se cubre temas sobre la arquitectura en la que funciona Neo4j y sus características.

- **Alta disponibilidad**

La versión de alta disponibilidad de Neo4j viene contenida en la edición Enterprise dentro del módulo Neo4j HA, que permite ejecutar la base de datos en arquitectura de clúster. Neo4j usa el tipo de replicación maestro-esclavo, además ofrece dos características, resiliencia y tolerancia a fallos, como se ejemplifica en la figura 2.41.

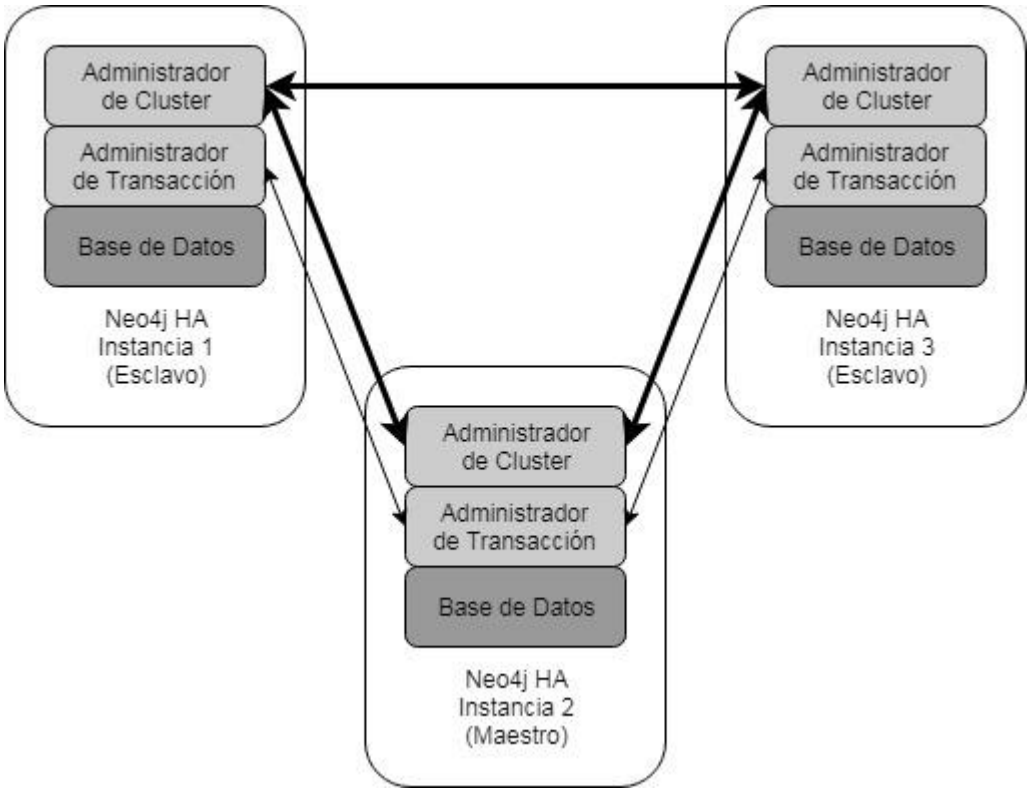


Figura 2.41 Replicación en Neo4j

En muchas ocasiones las bases de datos deben continuar funcionando a pesar de presentar fallos en la conectividad de red o si un nodo falla, esta deberá continuar trabajando, estos escenarios se le conoce como resiliencia y tolerancia a fallos.

La escalabilidad en Neo4j se realiza de forma horizontal, su propósito es agilizar la lectura intensiva en los nodos esclavos.

A diferencia de la arquitectura típica de maestro-esclavo, donde solo se puede escribir en el nodo maestro y leer desde los nodos esclavos, en Neo4j se pude escribir en cualquier nodo, cuando se realiza una escritura en algún nodo esclavo, primero se realiza la replicación de los datos al nodo maestro y este posteriormente se encarga de distribuir los datos a los demás nodos esclavos, los eventos de actualización de replicar los datos en los nodos son eventualmente consistentes, debido a que no se realizan instantáneamente, ni están disponibles al momento. La descripción grafica se puede observar en la figura 2.42 y en la figura 2.43.

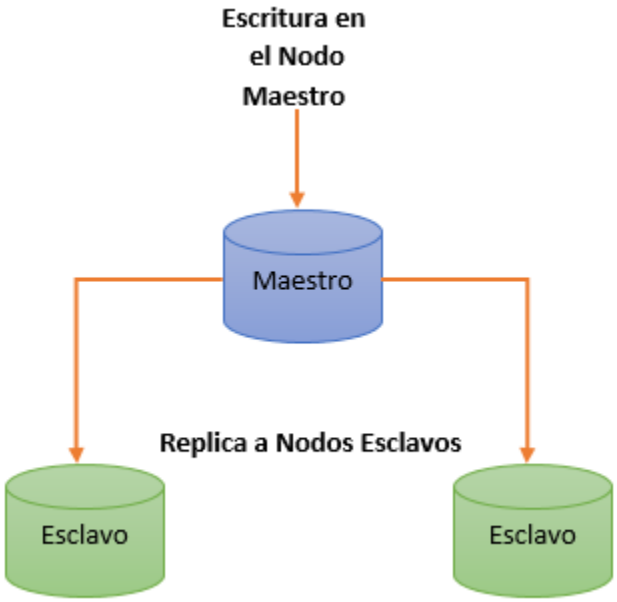


Figura 2.42 Replicación con escritura en el nodo maestro

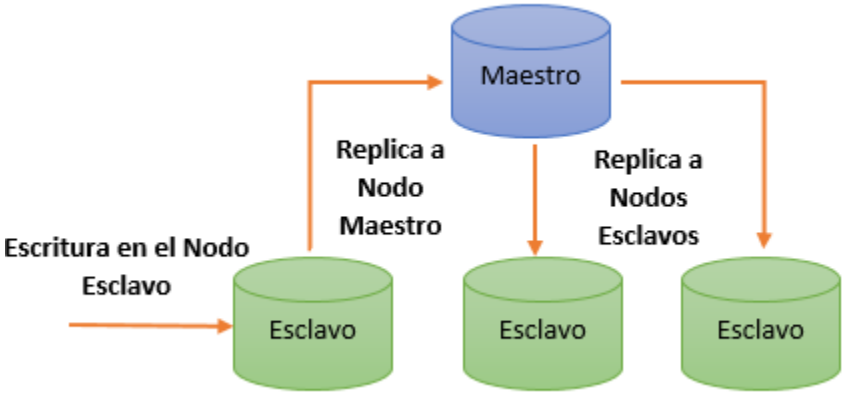


Figura 2.43 Replicación con escritura en el nodo esclavo

Capítulo 3
Análisis, diseño y arquitectura del sistema

3. Análisis, diseño y arquitectura del sistema

En este capítulo se realizará un análisis de requerimientos, el diseño y la arquitectura del sistema de punto de venta para restaurantes. Se seguirá el enfoque del desarrollo web MEAN-STACK que es derivado del desarrollo FULL-STACK, el cual consiste en construir aplicaciones complejas que involucra desarrollar componentes del lado del servidor (base de datos, número de conexiones, tiempo de duración de las sesiones, etc.) como del lado del cliente (presentación de la información, formularios, validación de datos, etc.).

Para realizar este tipo de desarrollo, MEAN utiliza el conjunto de tecnologías y frameworks tales como: MongoDB, Express, AngularJS y NodeJS. Cada uno de estos elementos se enfoca en el desarrollo del lado del cliente o del servidor. El desarrollo web MEAN-STACK está basado en el lenguaje de programación JavaScript y en el formato de archivos JSON, lo que hace que el desarrollo se más ágil.

3.1 Requerimientos y modelo del negocio

Se requiere de un sistema de punto de venta para administrar restaurantes de bajo costo. Esto implica que la aplicación deberá operar desde cualquier máquina o dispositivo de bajos recursos, sin necesidad de hacer uso de hardware especializado y costoso. La aplicación está dirigida a PyMES.

El principal objetivo de la aplicación es agilizar y optimizar la administración del negocio, de tal forma que los usuarios finales tengan la autonomía de ingresar información de una forma ágil y rápida. Por ejemplo, captura de pedidos a surtir en las diferentes estaciones de servicio (barra de bebidas, área de cocina, etc.).

La aplicación tendrá la capacidad de escalar, de forma vertical, incrementando la capacidad de procesamiento al servidor (memoria, CPU, espacio en disco, etc.) y de forma horizontal, agregando una mayor cantidad de servidores para distribuir la carga de trabajo. Estas modificaciones no deberán de afectar a las funcionalidades ya existentes.

El sistema tendrá la capacidad de modificar y agregar un nuevo modelo de negocio, como incluir nuevos esquemas de datos, implementar módulos de inventarios, seguimiento de ventas, análisis y minería de datos, conexión a plataformas financieras, etc.

El sistema también deberá de cubrir las siguientes características:

- Seguridad y Autenticación. Cada persona que ingrese al sistema deberá ingresar por medio de una contraseña única. Se omite el uso del usuario, esto con la finalidad de ingresar con mayor rapidez a la aplicación de aquí la justificación para restringir el uso de contraseñas únicas.

- Patrón de arquitectura de software por capas, para hacer más entendible y darle mantenimiento al código fuente estará separado por:
    - Capa de presentación
    - Capa de negocio
    - Capa de base de datos

- Nivel de acceso por perfiles, para ingresar al sistema se considera tres tipos de perfiles:
    - Mesero
    - Cajero
    - Administrador

- La aplicación funcionará bajo el protocolo HTTP, su interfaz será mediante una página web.
- Manejo concurrente de usuarios, varios usuarios podrán acceder al mismo tiempo a la aplicación.
- Los datos se mostrarán en tiempo real. Cualquier modificación que se realice sobre los datos, el sistema reflejará inmediatamente los cambios a todos los usuarios que soliciten la información.
- La interfaz será responsiva, ajustable a distintos tamaños de pantalla de los diferentes dispositivos con los que se accede a la aplicación.
- Por motivos de rapidez y seguridad, el sistema debe trabajar en una red local de alta velocidad.
- El sistema debe soportar la configuración en red de varias impresoras para la impresión y distribución de tickets en su correspondiente estación de servicio (barra de bebidas, área de cocina, etc.).

- Perfil Mesero
    - Será encargado de crear cuentas nuevas y levantar pedidos.

- Al crear una cuenta nueva, se debe generar automáticamente un folio para la comanda, este folio estará compuesto por la hora y fecha actual, concatenado con el usuario del mesero.
- Para que el mesero pueda abrir una nueva cuenta, debe ingresar número de mesa y número de clientes.
- Dentro de la interfaz gráfica, debe existir una sección para realizar consultas de todos los productos existentes en el negocio, para que el mesero elija cual producto ingresar al pedido.
- En caso de existir un error al momento de capturar el pedido, el mesero puede eliminar los pedidos, siempre y cuando no este confirmado el pedido, es decir, que aún no se haya impreso el ticket del pedido, ni surtido en la estación de trabajo.
- Los meseros pueden ver solamente las cuentas del día actual y únicamente pueden trabajar sobre sus propias cuentas, por ningún motivo pueden ver las cuentas de otros meseros ni de días pasados.
- Una vez que el mesero este seguro de los productos que va ingresar en el pedido, así como la cantidad de cada uno de ellos, se deberá confirmar los pedidos imprimiendo el ticket del pedido en su correspondiente estación de servicio (barra de bebidas, cocina, etc.).
- Los productos deben tener una clasificación para saber en qué estación de servicio van a ser surtidos, por ejemplo, refrescos, limonadas y aguas, deben pertenecer a la categoría de barra de bebidas.
- Cada vez que se imprima un ticket de pedido, este deberá de agrupar todos los productos correspondientes a la misma categoría y descartar todos los productos que ya fueron surtidos o enviados a imprimir anteriormente.

  Se deberá imprimir un ticket por mesero y contendrá datos relevantes como, nombre de mesero, categoría a la que pertenece el pedido, folio de la comanda, hora, folio del consecutivo del pedido, cantidad de productos, nombre de producto, observaciones por producto y número de mesa.

La figura 3.1 muestra el ejemplo de ticket:

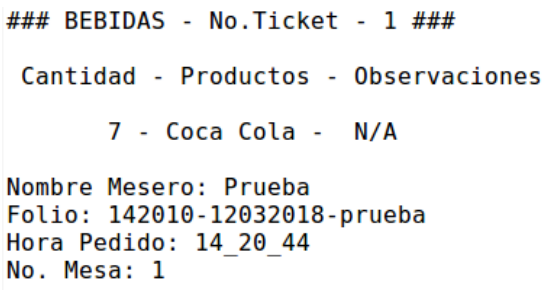


Figura 3.1 Muestra de ticket

- El ticket del pedido deberá contener un folio consecutivo, sin importar que mesero ingrese el pedido. Este consecutivo se incrementará en 1 por cada ticket impreso, se reiniciará automáticamente la secuencia de los folios cada día, cada estación de servicio (barra de bebidas, cocina, etc.) tendrá su propia secuencia de folios.

  En la figura 3.2 se indica el folio de los tickets:

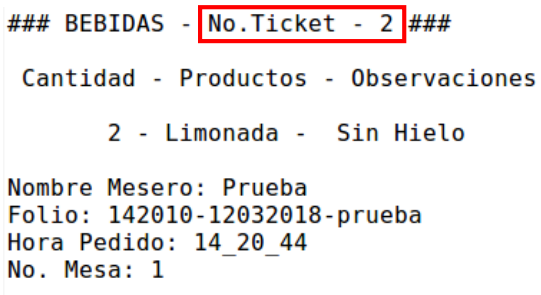


Figura 3.2 Folio de ticket

- Deberá existir una sección en la cual el mesero pueda consultar las cuentas que tiene abiertas.

- El sistema tendrá la capacidad de agregar productos a cuentas existentes, siempre y cuando la cuenta no esté cerrada o cobrada, los productos agregados deberán persistir en la base de datos y ser recuperados cada vez que el mesero requiera consultarlos.

- El mesero podrá imprimir el ticket con el total de la cuenta, para que se entregue al cliente para su posterior pago.

- El ticket del total de la cuenta mostrará información como, datos del establecimiento en el encabezado (nombre del negocio, dirección, datos fiscales, etc.), cantidad de productos, precio de los productos, nombre del producto, importe, total de la cuenta, en caso de aplicarse un descuento también se deberá mostrar el monto del descuento, nombre del mesero, fecha, hora, número de mesa, número de clientes y folio de la comanda.

- Cuando el mesero entregue el dinero del monto total de la cuenta al cajero, este deberá cobrar y cerrar la cuenta; en cuanto se cierra la cuenta, esta se bloquea y ya no se puede agregar más productos o imprimir tickets.

- Perfil Cajero
  - Este tipo de usuario puede cobrar y cerrar cuentas. Cuando el mesero entrega el monto total de la cuenta al cajero, este deberá buscar en el sistema, la cuenta a cobrar. En esta sección se mostrarán todas las cuentas ordenadas. En la parte superior se posicionan las cuentas que aún no han sido cobradas y que siguen abiertas y en la parte inferior todas las cuentas que ya fueron cobras y cerradas, el cajero deberá seleccionar la cuenta para cobrarla e indicarle al sistema que ya se cobró, es cuando el sistema bloquea automáticamente la cuenta para que ya no se pueda modificar.

  - Aplicar descuentos. Si por alguna razón se requiere aplicar un descuento a una cuenta, el cajero puede aplicarlo, las condiciones para aplicar un descuento son las siguientes:
    - El descuento no puede ser mayor a la cantidad del monto total de la cuenta.
    - Solo se puede aplicar una sola vez el descuento, por lo que no se permite aplicar varios descuentos a la misma cuenta.
    - El descuento se mostrará en el ticket impreso del total de la cuenta, tendrá la etiqueta "`Descuento: $ Cantidad`"

  - Re-imprimir pedidos, en caso de que se extravié un ticket de los pedidos, el cajero puede re-imprimir los pedidos, solo se tendría que seleccionar la estación de servicio y la fecha para mostrar en pantalla todos los pedidos que han

ingresado los meseros. Al contar con un folio consecutivo en cada ticket de los pedidos, basta con identificar el folio extraviado para re-imprimirlo.

- Perfil Administrador
    - Consultar cuentas de los meseros. El administrador podrá consultar todas las cuentas que fueron creadas durante el trascurso del día. Las cuentas se mostrarán ordenadas de forma descendiente, en la parte superior estarán las cuentas no cobradas o abiertas y posteriormente las cuentas que ya fueron cerradas o cobrada. Las cuentas abiertas estarán marcadas con un color que represente un mensaje de alerta y con un color diferente las cuentas que ya han sido cobradas y cerradas satisfactoriamente.

      También la búsqueda se podrá realizar por fecha.

    - Cobrar cuentas. Al igual que el perfil del cajero el administrador también tiene la facultad de cobrar cuentas.

    - Eliminar cuenta. Las cuentas se podrán eliminar solo si el monto total es igual a cero. Por ningún motivo se pueden eliminar cuentas que tengan una cantidad diferente a cero, se tiene que guardar un histórico de todas las cuentas que ya han sido atendidas para que se vean reflejadas en los reportes.

    - Dar de alta empleados nuevos. Por medio del sistema se muestra una interfaz para dar de alta nuevos empleados, en donde se solicitan datos como, nombre, apellidos, calle, número, colonia, delegación o municipio, estado, edad, número telefónico de contacto, e-mail, CURP, fecha de nacimiento, fecha de ingreso, género, puesto o perfil de acceso al sistema, indicar si el usuario puede acceder de inmediato al sistema o permanecerá bloqueado, id de usuario y contraseña.

    - El id del usuario, se generará automáticamente por el sistema, estará conformado por las letras del nombre y los apellidos, concatenado y sin espacios para garantizar que el id de usuario sea único en el sistema. En caso de existir nombres iguales, el sistema indicará que ya existe el usuario y manualmente se le podrá agregar un identificar al usuario en conflicto.

- Consultar empleados, en esta sección se podrá consultar toda la información acerca de los usuarios que están dados de alta en el sistema, los usuarios estarán ordenados, en la parte superior los usuarios que están habilitados para usar el sistema y en la parte inferior los usuario bloqueados.

- Modificar y eliminar empleados. En este apartado se podrá actualizar los datos de contacto y la contraseña, si el usuario olvidó su contraseña o necesita que se le asigne otra contraseña por motivos de seguridad, en esta sección se podrá realizar este cambio, también se podrá modificar únicamente los datos de contacto del usuario, como su número telefónico y mail, por último, en caso de ser requerido eliminar por completo al usuario de sistema.

- Dar de alta productos nuevos. En esta sección se requiere que el sistema solicite datos como: nombre del producto, precio, estación de servicio y categoría a la que pertenece el producto, por ejemplo, un refresco, pertenece a la estación de servicio de barra de bebidas y entra en la categoría de bebidas embotelladas.

- Consultar productos, aquí se muestran todos los productos existentes en el sistema ordenados por categoría.

- Modificar y eliminar productos, realizar modificación en los datos de un producto, tales como: nombre del producto, precio, estación de servicio o la categoría a la que pertenece. También se puede eliminar por completo del sistema el producto seleccionado.

- Reportes, se despliega una breve información sobre las ventas reportadas en el sistema, para generar el reporte, se consideran cuentas cobradas como cuentas no cobradas, el reporte se muestra desplegando las siguientes secciones:

    - Ventas totales del día
    - Ventas por mesero
    - Cantidad cobrada por cajeros
    - Descuentos aplicados
    - Top productos más vendidos durante el día

3.2 Diseño del sistema

El sistema se implementa bajo la metodología MEAN-STACK, que como ya se explicó al principio del capítulo, se refiere al desarrollo que involucra la construcción de módulos de software tanto del lado del servidor como del lado del cliente, básicamente, consiste en generar instrucciones mediante la escritura de código fuente para que el sistema sea capaz de recibir y procesar peticiones de los usuarios por medio de la interfaz gráfica o técnicamente conocido como front end.

Posteriormente la solicitud lanza un evento al servidor de NodeJS, quien captura el evento y lo envía al loop de eventos en espera de ser resuelto. Si el evento requiere de una acción en base de datos, entonces la solicitud es procesada por el framework de Express que a su vez hará uso de la librería Mongoose (el funcionamiento de Express y Mongoose se explicará más a detalle en subtemas posteriores), para recuperar los datos de MongoDB y devolver la respuesta al loop de eventos. Cuando el evento finaliza, ya sea de forma exitosa o en error, se envía la notificación a NodeJS quien muestra el resultado por medio de AngularJS.

En la figura 3.3, se muestra el flujo del funcionamiento de Node.js con la metodología Mean-Stack.

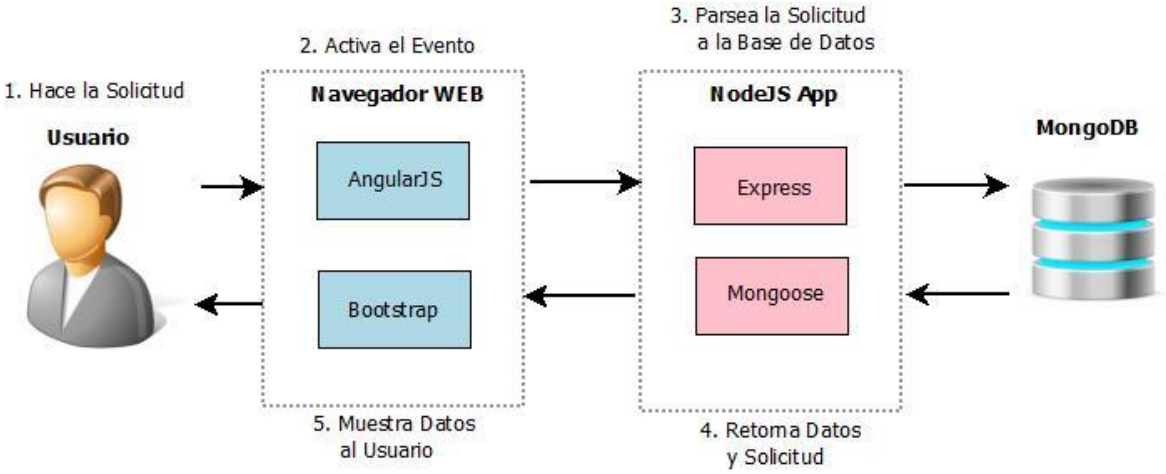


Figura 3.3 Flujo metodología Mean-Stack

El desarrollo es multicapa, se separa el código fuentes en 3 categorías: la capa de presentación, capa de negocio y capa de acceso a datos, la finalidad es organizar el código

fuente, obtener un bajo acoplamiento entre los módulos para minimizar los cambios en el sistema cuando se requiera modificar un módulo y generar módulos con una alta cohesión para que cada módulo cumpla únicamente su propósito para el cual fue desarrollado.

La capa de presentación hace referencia a todos los objetos con los que el usuario interactúa, ya sea para visualizar o modificar datos, tales como formularios, tablas, menús, botones, cuadros de texto, etc.

La capa de negocio estará representada por el código que implementa las reglas del negocio y será la encargada de validar los datos que sean los esperados por el sistema, para evitar contingencias.

Por último, la capa de acceso a datos es la que se encarga de persistir los objetos dentro de la base de datos.

A continuación se detallan las funcionalidades de cada módulo del sistema por tipo de usuario y su correspondiente adaptación a la metodología seleccionada.

3.2.1 Diseño de módulos y funcionalidades

De acuerdo a los requerimientos establecidos en la sección 3.1, se realizan los diagramas de caso de uso para cada tipo de usuario.

3.2.1.1 Módulo Mesero

Es el perfil más bajo, su nivel de restricción es muy alto, solo puede acceder a los datos que el mismo genera. A continuación en la figura 3.4 se muestra el diagrama de casos de uso del perfil de mesero.

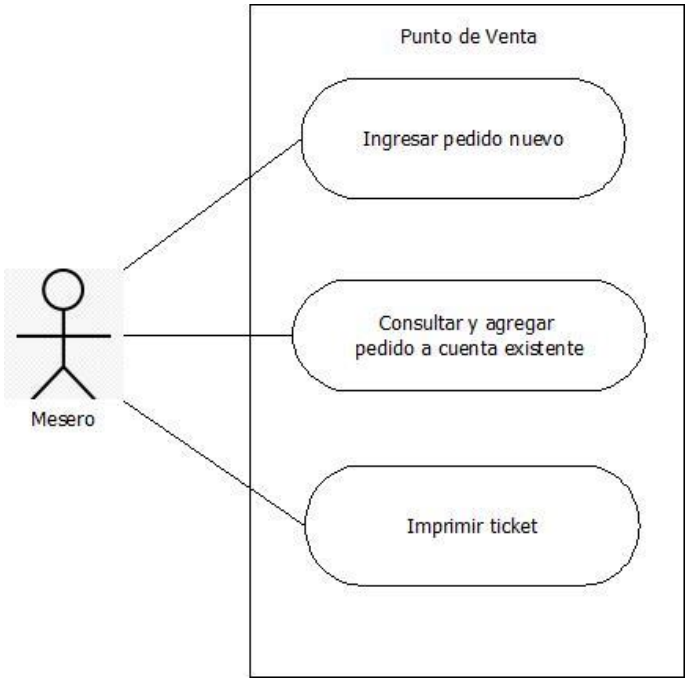


Figura 3.4 Casos de uso Mesero

Descripción pedido nuevo	
Nombre:	Ingresar pedido nuevo
Actores:	Mesero
Función:	Permitir abrir una cuenta nueva
Descripción:	El mesero puede generar una nueva cuenta para agregar pedidos y que estos sean surtidos en las diferentes estaciones de trabajo.
Referencia:	Figura 3.4

Tabla 3.5 Descripción de casos de uso pedido nuevo.

Descripción consulta y agregar pedidos	
Nombre:	Consultar y agregar pedido a cuenta existente
Actores:	Mesero
Función:	Permitir consultar y agregar un pedido a una cuenta existente
Descripción:	En la sección de búsqueda, el mesero puede visualizar todas las cuentas que ha creado durante el transcurso del día. El mesero agrega pedidos a una cuenta que ya fue creada anteriormente.
Referencia:	Figura 3.4

Tabla 3.6 Descripción de casos de consultar y agregar pedidos.

Descripción imprimir ticket	
Nombre:	Imprimir ticket
Actores:	Mesero
Función:	Permitir imprimir ticket del total de la cuenta
Descripción:	El mesero tiene la facultad de imprimir el ticket de la cuenta total.
Referencia:	Figura 3.4

Tabla 3.7 Descripción de casos de imprimir ticket.

3.2.1.2 Módulo Cajero

Este tipo de perfil tiene un nivel intermedio de acceso a la aplicación, el usuario no genera información, sin embargo, tiene la capacidad de ver la información generada por los meseros, se encarga de monitorear y manipular la información. En la figura 3.8 se muestra el diagrama de casos de uso para el perfil de cajero

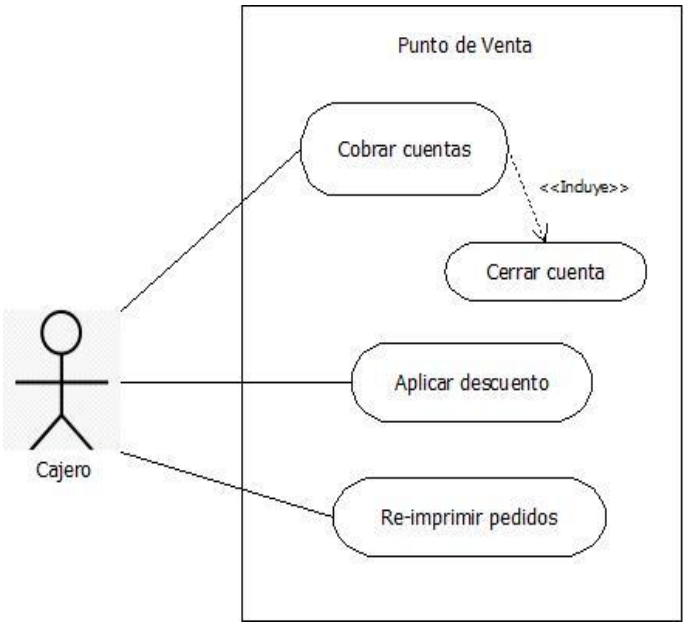


Figura 3.8 Casos de uso Cajero

Descripción cobrar cuenta	
Nombre:	Cobrar cuentas
Actores:	Cajero
Función:	Permitir cobrar una cuenta
Descripción:	En cuanto el mesero haya entregado el dinero, el cajero procede a cobrar la cuenta. Si la cuenta tiene pedidos pendientes por confirmar, entonces la cuenta no puede ser cobrada. Si la cuenta esta en ceros tampoco se puede cobrar. En cuanto se cobra la cuenta, esta se cierra y se bloquea en automático para evitar agregar más productos.
Referencia:	Figura 3.8

Tabla 3.9 Descripción de casos de cobrar cuenta.

Descripción descuentos	
Nombre:	Aplicar descuento
Actores:	Cajero
Función:	Permitir aplicar un descuento
Descripción:	El cajero puede aplicar descuentos justificados. No se puede aplicar descuentos mayores al monto total de la cuenta. Tampoco se permite aplicar descuentos en cuentas que tengan pedidos pendientes por confirmar. Únicamente se aplica un descuento por cuenta.
Referencia:	Figura 3.8

Tabla 3.10 Descripción de casos de descuentos.

Descripción re-imprimir ticket pedido	
Nombre:	Re-imprimir pedidos
Actores:	Cajero
Función:	Permitir re-imprimir tickets de los pedidos
Descripción:	El cajero puede re-imprimir los tickets de los pedidos, en una sección de búsqueda localiza el ticket que quiere imprimir, se agrupa por estación de servicio y por fecha.
Referencia:	Figura 3.8

Tabla 3.11 Descripción de casos de re-imprimir ticket pedido.

3.2.1.3 Módulo Administrador

El perfil del administrador tiene acceso total a la información y puede modificar toda clase de información. En la figura 3.12 se muestra el diagrama de casos de uso para el perfil del administrador.

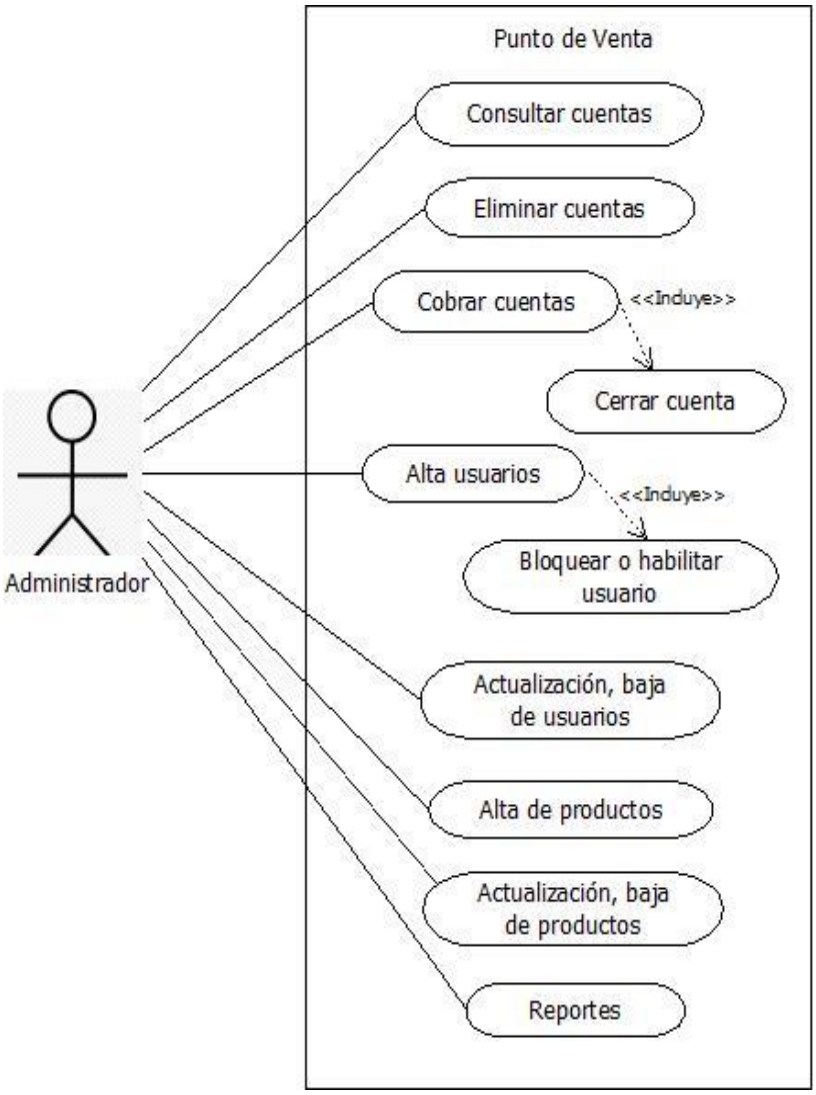


Figura 3.12 Casos de uso Administrador

Descripción consultar cuentas	
Nombre:	Consultar cuentas
Actores:	Administrador
Función:	Permitir consultar cuentas tanto abiertas como cerradas
Descripción:	Con esta funcionalidad se puede consultar que cuentas ya están cobradas y cuales falta por cobrar, están identificadas con un color diferente cada tipo de cuenta. Las cuentas deben estar ordenadas, en la parte superior las cuentas abiertas y en la parte inferior las cuentas cerradas.
Referencia:	Figura 3.12

Tabla 3.13 Descripción de casos de consultar cuentas.

Descripción eliminar cuentas	
Nombre:	Eliminar cuentas
Actores:	Administrador
Función:	Permitir eliminar cuentas
Descripción:	En esta sección se permite eliminar cuentas siempre y cuando el monto sea igual a cero.
Referencia:	Figura 3.12

Tabla 3.14 Descripción de casos de eliminar cuentas.

Descripción cobrar cuentas	
Nombre:	Cobrar cuentas
Actores:	Administrador
Función:	Permitir cobrar una cuenta
Descripción:	En cuanto el mesero haya entregado el dinero, el administrador puede cobrar la cuenta. Si la cuenta tiene pedidos pendientes por confirmar la cuenta no puede ser cobrada. Si la cuenta esta en ceros tampoco se puede cobrar. En cuanto se cobra la cuenta, esta se cierra y se bloquea en automático para evitar agregar más productos.
Referencia:	Figura 3.12

Tabla 3.15 Descripción de casos de cobrar cuentas.

Descripción alta usuarios	
Nombre:	Alta usuarios
Actores:	Administrador
Función:	Permitir dar de alta un usuario
Descripción:	El administrador podrá dar de alta un usuario nuevo. Al momento de dar de alta se podrá elegir si habilitar o bloquear al usuario. El id del usuario se genera automáticamente concatenando el nombre y los apellidos.
Referencia:	Figura 3.12

Tabla 3.16 Descripción de casos de alta usuarios.

Descripción actualización y baja usuarios	
Nombre:	Actualización, baja de usuarios
Actores:	Administrador
Función:	Permitir actualizar y dar de baja a los usuarios
Descripción:	Únicamente se puede actualizar la contraseña y los datos de contacto del usuario. Se puede eliminar por completo al usuario del sistema.
Referencia:	Figura 3.12

Tabla 3.17 Descripción de casos de actualización y baja de usuarios.

Descripción alta productos	
Nombre:	Alta de productos
Actores:	Administrador
Función:	Permitir dar de alta nuevos productos en el sistema
Descripción:	En esta sección se puede agregar un producto nuevo al sistema, donde se tiene que indicar la clasificación, tipo de estación de servicio y precio.
Referencia:	Figura 3.12

Tabla 3.18 Descripción de casos de alta productos.

Descripción actualización y baja productos	
Nombre:	Actualización, baja de productos
Actores:	Administrador
Función:	Permitir actualizar y dar de baja productos del sistema
Descripción:	Se puede actualizar todos los datos del producto y borrarlo por completo del sistema
Referencia:	Figura 3.12

Tabla 3.19 Descripción de casos de actualización y baja de productos.

Descripción reportes	
Nombre:	Reportes
Actores:	Administrador
Función:	Permitir generar reportes por día
Descripción:	En esta sección se visualiza las ventas totales y datos como, ventas por meseros, top de productos más vendidos y total de descuentos aplicados.
Referencia:	Figura 3.12

Tabla 3.20 Descripción de casos de reportes.

3.2.2 Elección e implementación de la base de datos NoSQL

Entre las razones del porque se eligió MongoDB, se debe principalmente a la capacidad de escalar los datos, debido a que no tiene una estructura estática en el esquema, por lo que puede ser dinámico. Dependiendo de las necesidades del negocio se puede alterar la estructura de los datos quitando o agregando campos a los documentos. En el enfoque de bases de datos relacionales es similar a agregar o quitar campos de una tabla.

El almacenamiento de datos en MongoDB se basa en archivos con formato JSON, el cual es sencillo y fácil de utilizar. Este tipo de formato se suele utilizar para almacenar información no estructurada, no se requiere de módulos intermediarios de software para realizar la integración entre MongoDB, Node.js y Angular.js, ya que todos utilizan o tienen la capacidad de trabajar con el formato JSON. Esta correspondencia facilita y agiliza la comunicación entre las distintas capas de la arquitectura de la aplicación.

Una de las características fundamentales de MongoDB es la replicación de los datos que permite el escalamiento horizontal. La técnica más común es conocida como "*sharding*". Esta técnica permite distribuir los datos de forma automática entre los distintos servidores asignados para el almacenamiento. Aunque de momento no es necesario ocupar este tipo de replicación, esta característica ofrece flexibilidad al sistema para que en un futuro pueda ser escalable.

Al no existir el concepto de tabla, existe lo que se conoce como colección en MongoDB. De acuerdo a las necesidades del negocio se crea el esquema de la figura 3.21, para almacenar la información y la relación que existe entre cada una de las colecciones.

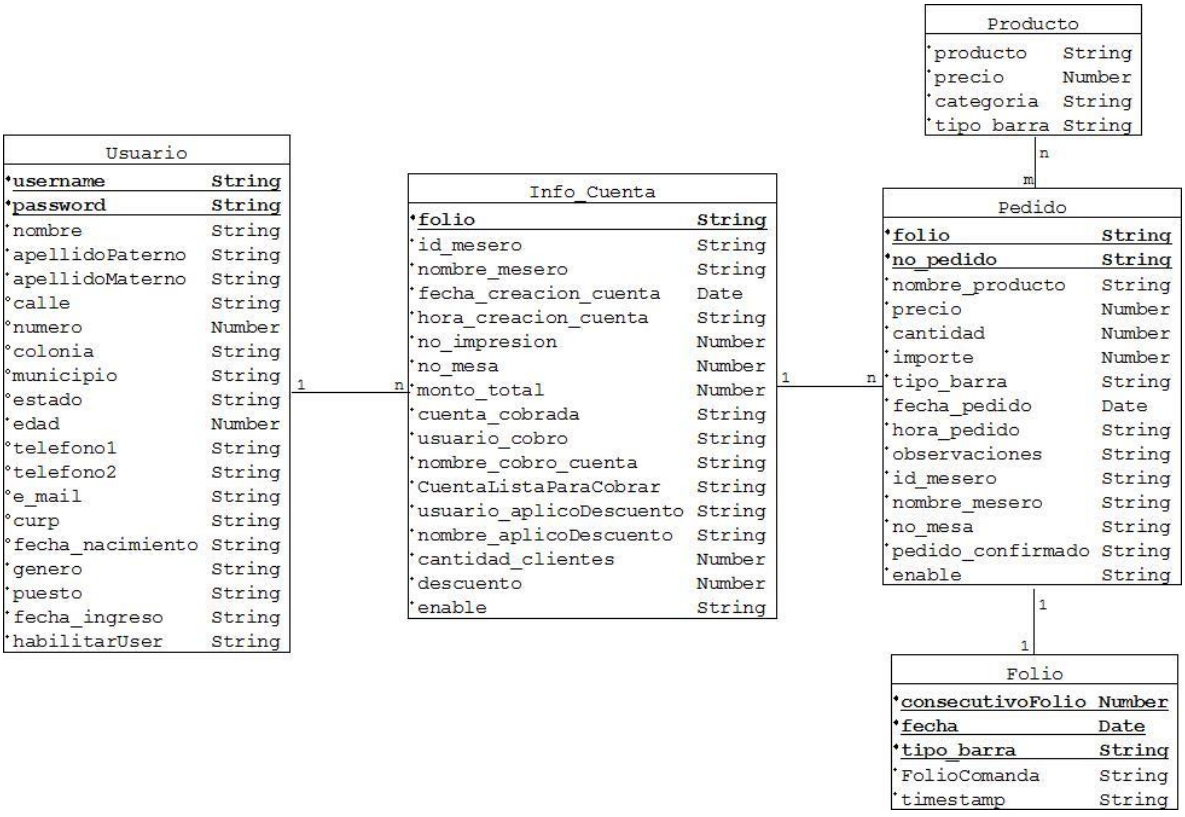


Figura 3.21 Esquema del Punto de Venta

No es necesario crear manualmente las colecciones para que el sistema funcione, el propio framework se encarga de su creación, solo basta con realizar la configuración de los esquemas con la librería Mongoose. A continuación se muestra el código fuente de cómo se realizó la creación de los esquemas en formato de JSON con mongoose.

Esquema Usuario:

```
var mongoose = require('mongoose');
var userSchema = mongoose.Schema({
          nombre: String,
          apellidoPaterno: String,
          apellidoMaterno: String,
          calle: String,
          numero:String,
          colonia: String,
          municipio: String,
          estado: String,
          edad: String,
          telefono1: String,
          telefono2: String,
          e_mail: String,
          curp: String,
          fecha_nacimiento: String,
          genero: String,
          puesto: String,
          fecha_ingreso: String,
          habilitarUser: String,
          username: String,
          password: String
});
module.exports = mongoose.model('User', userSchema);
```

Esquema Producto:

```
var mongoose = require('mongoose');
var productoSchema = mongoose.Schema({
     producto: String,
     precio: Number,
     categoria: String,
     tipo_barra: String
});
module.exports = mongoose.model('Productos', productoSchema);
```

Esquema Folio:

```
var mongoose = require('mongoose');
var folioSchema = mongoose.Schema({
     consecutivoFolio:Number,
     fecha: Date,
     FolioComanda: String,
     timestamp: String,
     tipo_barra: String
});
module.exports = mongoose.model('Folios', folioSchema);
```

Esquema Info_Cuenta:

```
var mongoose = require('mongoose');
var infoCuentaSchema = mongoose.Schema({
      folio: String,
      id_mesero: String,
      nombre_mesero: String,
      fecha_creacion_cuenta: Date,
      hora_creacion_cuenta: String,
      no_impresion: Number,
      no_mesa: Number,
      monto_total: Number,
      cuenta_cobrada: String,
      usuario_cobro: String,
      nombre_cobro_cuenta: String,
      CuentaListaParaCobrar:String,
      usuario_aplicoDescuento:String,
      nombre_aplicoDescuento:String,
      cantidad_clientes:Number,
      descuento:Number,
      enable:String
});
module.exports = mongoose.model('info_cuenta',
infoCuentaSchema);
```

Esquema Pedido:

```
var mongoose = require('mongoose');
var pedidoSchema = mongoose.Schema({
      folio: String,
      no_pedido:String,
      nombre_producto: String,
      precio: Number,
      cantidad: Number,
      importe: Number,
      tipo_barra: String,
      fecha_pedido: Date,
      hora_pedido: String,
      observaciones: String,
      id_mesero: String,
      nombre_mesero: String,
      no_mesa: String,
      pedido_confirmado: String,
      enable: String
});
module.exports = mongoose.model('pedidos', pedidoSchema);
```

El esquema mostrado es relativo, este puede cambiar de acuerdo a las necesidades del negocio. Para este caso, solo cumple con los propósitos establecidos, sin embargo, se puede agregar o quitar atributos o colecciones completas, todo esto sin la necesidad de que la aplicación sufra un cambio importante.

3.2.3 Elección e implementación del framework asíncrono

Se eligió Node.js como framework para la construcción del sistema, por varias razones.

1. Es un framework ligero y portable, funciona bajo las plataformas de Windows, Linux, Unix y Mac.
2. Tiene un gran rendimiento para aplicaciones en red de gran escala.

3. El lenguaje que se utiliza para construir aplicaciones es JavaScript, por lo que resulta bastante sencillo aprender el lenguaje y empezar a construir aplicaciones de gran calidad.
4. Permite conexiones concurrentes de miles de usuarios al mismo tiempo, disminuyendo la necesidad de implementar más servidores para soportar la carga de trabajo, caso contrario ocurre con otras aplicaciones desarrolladas en otros lenguajes.
5. No genera hilos por cada solicitud de conexión, solo genera un solo subproceso, manejo más eficiente de la memoria
6. Al utilizar el lenguaje JavaScript tanto del lado del cliente como del servidor, hace que se más rápida la comunicación de los datos.
7. Node.js posee un gestor de paquetes llamado NPM, donde existe una gran variedad de aplicaciones desarrolladas por una comunidad de programadores.
8. La respuesta de las solicitudes es en tiempo real. Tiene una gran agilidad para resolver los eventos lanzados.

Montar un servidor básico de Node.js es muy sencillo, basta con generar un archivo de configuración con la extensión .js y ejecutarlo junto con el comando *node* (previamente ya tiene que estar instalado Node.js en el servidor). Por ejemplo: el archivo *server.js* mostrado en la figura 3.22, tiene la siguiente configuración, para crear una instancia del servidor Node.js

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 8080;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Node.js en funcionamiento!\n');
});

server.listen(port, hostname, () => {
  console.log(`Ejecutando servidor: //${hostname}:${port}/`);
});
```

Figura 3.22 Código en JavaScript para configurar servidor Node.js

Una vez que ya se tiene preparado el archivo `server.js` se ejecuta Node.js, para generar una instancia de la aplicación, en la figura 3.23 y 3.24 se indica los comandos para levantar la instancia de Node.js

```
$ node server.js
```

Figura 3.23 Comando para ejecutar servidor Node.js

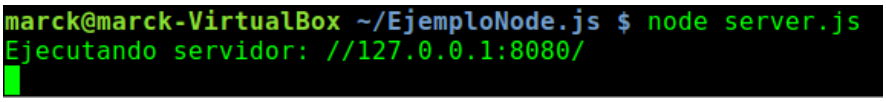


Figura 3.24 Generando instancia de Node.js

Si abrimos el explorador web como se observa en la figura 3.25 y accedemos a la dirección `127.0.0.1:8080` nos mostrará que el servidor de Node.js se encuentra funcionando:

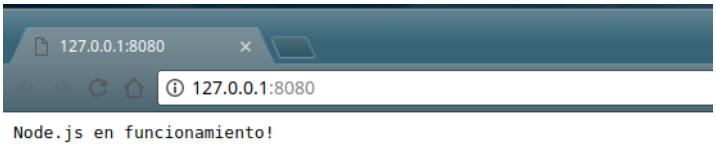


Figura 3.25 Explorador web, respuesta Node.js

La simplicidad de trabajar con Node.js facilita el desarrollo de software y reduce los tiempos de construcción. Aunque esto es solo un ejemplo demostrativo, la aplicación de Punto de Venta parte de estos principios fundamentales para iniciar el servidor, en la realidad lo que hace que funcione correctamente el sistema es el conjunto de librerías y módulos, que a continuación se explican.

3.2.4 NPM y dependencias

En Node.js existe una gran variedad de paquetes y módulos escritos en JavaScript por una comunidad de programadores, su propósito es facilitar el desarrollo de sistemas, ya que no es necesario volver a reescribir módulos que ya fueron escritos anteriormente y que cumplen un propósito específico. Por ejemplo, módulos para clientes HTTP, módulos de encriptado, módulos de testing, presentación y visualización de datos, analizadores de código, generar gráficas, etc.

Para obtener estos paquetes se requiere de NPM (package manager for JavaScript) quien se conecta a un repositorio central y se encarga de obtener paquetes y resolver dependencias de forma inteligente, por default se instala junto con Node.js.

Para instalar un módulo para encriptar la contraseña en la base de datos se tiene que seguir los siguientes pasos.

En la figura 3.26 se muestra como ejecutar el comando para instalar el módulo de bcryptjs.

```
marck@marck-VirtualBox ~ $ npm install bcryptjs
bcryptjs@2.4.3 node_modules/bcryptjs
```

Figura 3.26 Instalación de bcryptjs

Una vez que se realiza la instalación del módulo, para utilizarlo, dentro del código fuente de la aplicación se manda llamar de la siguiente forma como se indica en la figura 3.27:

```
var bcrypt = require('bcryptjs');
...
// Encripta Password
userSchema.methods.generateHash = function(password){
      return bcrypt.hashSync(password, bcrypt.genSaltSync(9));
}
// Des-encripta Password
userSchema.methods.validPassword = function(password){
      return bcrypt.compareSync(password, this.password);
}
```

Figura 3.27 Código JavaScript bcrypt

Así como este módulo se puede instalar cualquier otro módulo disponible en el repositorio. La gran mayoría tiene su propia documentación donde se explica cómo instalar y utilizar sus funciones.

3.2.5 Express

Es un framework enfocado a trabajar sobre aplicaciones web dinámicas, donde se requiere acceder a base de datos para realizar una acción. El framework determina qué tipo de acción se va realizar, dependiendo de la estructura de la URL el método http correspondiente: GET, POST, PUT o DELETE.

En Express también se configuran las rutas donde se almacenan las plantillas HTML para mostrar los resultados obtenidos. Igualmente se configura la gestión de cookies, sesiones y direccionamiento de rutas.

Para fines de este trabajo también se requiere instalar el módulo `body-parse`. La instalación se realiza junto con el middleware Express, se ejecuta el siguiente comando de la figura 3.28:

```
marck@marck-VirtualBox ~ $ npm install express body-parser
body-parser@1.18.3 node_modules/body-parser
├── content-type@1.0.4
├── bytes@3.0.0
├── depd@1.1.2
├── qs@6.5.2
├── on-finished@2.3.0 (ee-first@1.1.1)
├── raw-body@2.3.3 (unpipe@1.0.0)
├── debug@2.6.9 (ms@2.0.0)
├── http-errors@1.6.3 (setprototypeof@1.1.0, inherits@2.0.3,
statuses@1.5.0)
├── iconv-lite@0.4.23 (safer-buffer@2.1.2)
└── type-is@1.6.16 (media-typer@0.3.0, mime-types@2.1.19)

express@4.16.3 node_modules/express
├── escape-html@1.0.3
├── array-flatten@1.1.1
├── setprototypeof@1.1.0
├── cookie-signature@1.0.6
├── utils-merge@1.0.1
├── merge-descriptors@1.0.1
├── content-type@1.0.4
├── methods@1.1.2
├── encodeurl@1.0.2
```

```
├── parseurl@1.3.2
├── vary@1.1.2
├── range-parser@1.2.0
├── cookie@0.3.1
├── fresh@0.5.2
├── etag@1.8.1
├── content-disposition@0.5.2
├── path-to-regexp@0.1.7
├── serve-static@1.13.2
├── statuses@1.4.0
├── safe-buffer@5.1.1
├── depd@1.1.2
├── qs@6.5.1
├── on-finished@2.3.0 (ee-first@1.1.1)
├── finalhandler@1.1.1 (unpipe@1.0.0)
├── debug@2.6.9 (ms@2.0.0)
├── proxy-addr@2.0.3 (forwarded@0.1.2, ipaddr.js@1.6.0)
├── send@0.16.2 (destroy@1.0.4, ms@2.0.0, mime@1.4.1, http-
errors@1.6.3)
├── type-is@1.6.16 (media-typer@0.3.0, mime-types@2.1.19)
├── accepts@1.3.5 (negotiator@0.6.1, mime-types@2.1.19)
└── body-parser@1.18.2 (bytes@3.0.0, http-errors@1.6.3,
iconv-lite@0.4.19, raw-body@2.3.2)
```

Figura 3.28 Instalación de Express y Body-Parser

Una vez realizada la instalación de los módulos podemos usar el middleware que es el componente encargado del intercambio de información entre dos aplicaciones, así como se muestra en el ejemplo de la figura 3.29, donde se realiza una pequeña configuración para indicar la ruta de almacenamiento de los archivos HTML de las vistas y el puerto por el cual se reciben las solicitudes.

```
var express = require('express');
var app = express();
var port = process.env.PORT || 8080;
var bodyParser = require('body-parser');
app.configure(function() {
     app.use(express.bodyParser());
});
app.use(express.static(__dirname + '/views/'));
app.engine('html', require('ejs').renderFile);
app.listen(port);
console.log('Server running on port: ' + port);
```

Figura 3.29 Código para configurar Express

Express resuelve las solicitudes dependiendo de la ruta que se le pase como parámetro a la función para resolver la solicitud. La nomenclatura para este tipo de funciones se muestra en la figura 3.30.

```
app.put(path, callback [, callback ...])
app.post(path, callback [, callback ...])
app.get(path, callback [, callback ...])
app.delete(path, callback [, callback ...])
```

Figura 3.30 Nomenclatura de funciones para Express

En el ejemplo de la figura 3.31, se muestra como obtener datos desde la base de MongoDB con una solicitud GET.

```
var pedidos = require('./modelo/pedidos');

app.get('/api/get/pedidos', Pedidos.find({
            enable: 'Si'
        },
        function(err, pedidos) {
          if (err)
             res.send(err);
          else
             // devuelve todas los Pedidos en JSON
             res.json(pedidos);
        });
);
```

Figura 3.31 Código solicitud GET

En el ejemplo del método GET, se requiere especificar cuál es el modelo que se va utilizar, la ruta por la cual se va ingresar la solicitud para obtener los pedidos que existen en la base de datos MongoDB y que coincidan con el filtro de la función `find`, el resultado de la búsqueda se le indica que es en formato JSON y finalmente se muestran los datos en la plantilla HTML asignada.

3.2.6 Mongoose

Este módulo sirve para conectarse a una base de datos y crear esquemas a partir de modelos.

Para instalar mongoose se ejecuta el comando de la figura 3.32.

```
marck@marck-VirtualBox ~ $ npm install mongoose
mongoose@5.2.5 node_modules/mongoose
├── ms@2.0.0
├── mongoose-legacy-pluralize@1.0.2
├── lodash.get@4.4.2
├── sliced@1.0.1
├── regexp-clone@0.0.1
├── kareem@2.2.1
├── mpath@0.4.1
├── bson@1.0.9
├── mongodb@3.1.1
├── mongodb-core@3.1.0 (saslprep@1.0.1,
require_optional@1.0.1)
├── mquery@3.0.0 (sliced@0.0.5, debug@2.6.9, bluebird@3.5.0)
└── async@2.6.1 (lodash@4.17.10)
```

Figura 3.32 Instalación de mongoose

En el siguiente código de la figura 3.33 se muestra como realizar una conexión a la base de datos MongoDB y crear un nuevo esquema.

Archivo para conectarse a la base de datos: `server.js`

```
var mongoose = require('mongoose');
var configDB = require('./config/database.js');
mongoose.connect(configMongoDB.url);
```

Figura 3.33 Código conexión a una base de datos con mongoose

En la figura 3.34 se muestra el archivo de la URL de conexión: `configMongoDB.url`

```
module.exports = {
      'url' : 'mongodb://127.0.0.1:27017/MiBaseDeDatos'
}
```

Figura 3.34 URL de conexión a una base de datos con mongoose

En la figura 3.35 se muestra como crear el esquema del modelo de usuario: `user.js`

```
var mongoose = require('mongoose');
var userSchema = mongoose.Schema({
          nombre: String,
          apellidoPaterno: String,
          apellidoMaterno: String,
          calle: String,
          numero: Number,
          colonia: String,
          municipio: String,
          estado: String,
          edad: Number,
          telefono1: String,
          telefono2: String,
          e_mail: String,
          curp: String,
          fecha_nacimiento: String,
          genero: String,
          puesto: String,
          fecha_ingreso: String,
          habilitarUser: String,
          username: String,
          password: String
});
          module.exports = mongoose.model('User',
userSchema);
```

Figura 3.35 Modelo de datos con mongoose

3.2.7 Passport

Es un módulo de autentificación, enfocado a aplicaciones web que utilicen el módulo de Express. Permite métodos de autentificación ya sea locales o por medio de Facebook, Twitter y otras plataformas.

Para instalar Passport se utiliza el comando de la figura 3.36:

```
marck@marck-VirtualBox ~ $ npm install passport
passport@0.2.2 node_modules/passport
├── passport-strategy@1.0.0
└── pause@0.0.1
```

Figura 3.36 Instalación Passport

En el siguiente ejemplo se muestra como realizar un método de autentificación local, primero se realiza una búsqueda para identificar el id del usuario y posteriormente se realiza la validación del password encriptado.

En la figura 3.37 se muestra el archivo de configuración Passport: `passport.js`

```
var LocalStrategy = require('passport-local').Strategy;
var User = require('../app/modelo/user');
module.exports = function(passport) {
     passport.serializeUser(function(user, done){
     done(null, user.id);
     });
     passport.deserializeUser(function(id, done){
     User.findById(id, function(err, user){
     done(err, user);
     });
});
passport.use('local-login', new LocalStrategy({
          usernameField: 'email',
          passwordField: 'password',
          passReqToCallback: true
     },
     function(req, email, password, done){
          process.nextTick(function(){
User.findOne({'username': email, 'habilitarUser':'Si'},
function(err, user){
          if(err)
               return done(err);
          if(!user)
               return done(null, false,
req.flash('loginMessage', 'No se encontró el usuario o está
inhabilitado para usar el sistema, favor de consultar con el
Administrador'));
          if(!user.validPassword(password)){
               return done(null, false,
req.flash('loginMessage', 'Password Incorrecto'));
                         }
                return done(null, user);
               });
          });
     }
));
};
```

Figura 3.37 Configuración Passport

En la figura 3.38 se muestra el archivo del modelo de usuario: `user.js`

```
var mongoose = require('mongoose');
var bcrypt = require('bcryptjs');
var userSchema = mongoose.Schema({
          nombre: String,
          apellidoPaterno: String,
          apellidoMaterno: String,
          calle: String,
          numero:String,
          colonia: String,
          municipio: String,
          estado: String,
          edad: String,
          telefono1: String,
          telefono2: String,
          e_mail: String,
          curp: String,
          fecha_nacimiento: String,
          genero: String,
          puesto: String,
          fecha_ingreso: String,
          habilitarUser: String,
          username: String,
          password: String
});
userSchema.methods.generateHash = function(password){
     return bcrypt.hashSync(password, bcrypt.genSaltSync(9));
}
userSchema.methods.validPassword = function(password){
     return bcrypt.compareSync(password, this.password);
}
module.exports = mongoose.model('User', userSchema);
```

Figura 3.38 Modelo User

3.2.8 AngularJS y Bootstrap

Representan a 2 de los frameworks Javascript más populares ideales para construir la capa de presentación de la aplicación. Tanto Angular como Bootstrap poseen librerías que facilitan y simplifican el desarrollo de las vistas de una aplicación.

Para utilizar estos frameworks basta con importar el código fuente desde su página de origen, como se muestra en la figura 3.39.

```
<!doctype html>
<html ng-app>
  <head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.2/an
gular.min.js"></script>
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/boots
trap.min.js"></script>
  </head>
  <body>
    <div>
      <label>Nombre:</label>
     <input type="text" ng-model="Nombre" placeholder=
          "Escribe tu nombre" >
      <hr>
      <h1>Hola {{Nombre}}!</h1>
    </div>
  </body>
</html>
```

Figura 3.39 Archivo HTML con Angular.js

Gracias a la combinación de estos dos frameworks se puede aplicar estilos, mejorar la presentación de los objetos, dar formato a textos, validar formularios, aplicar filtros, ordenar datos, crear objetos de tipos "*scopes"* para ligarlos con datos que se extraen de la base de datos, etc.

3.3 Topología de red

La aplicación funciona en una red local inalámbrica, debido a que el tiempo de respuesta debe ser en tiempo real, evitar la segmentación de redes y pérdidas de paquetes de información. Los principales componentes de la aplicación son:

- Servidor (Computadora con Windows, Linux o Mac).
- Dispositivos de acceso a la aplicación (Tablets, Computadoras, Teléfonos celular con android, ios, windows, etc.)
- Router.
- Impresora(s) Térmica(s) de Tickets.

El sistema tiene la capacidad de configurar una o varias impresoras, mediante un archivo de configuración se dan de alta las impresoras indicando cuál es su correspondiente estación de trabajo.

3.3.1 Diagrama con una impresora de tickets

Para configurar una sola impresora de tickets, esta se configura por medio del puerto USB en el servidor de la aplicación, los dispositivos de acceso a la aplicación tendrán IP dinámica y el servidor de la aplicación tendrá IP estática. La figura 3.40 muestra la topología de red para una impresora de tickets.

Figura 3.40 Diagrama con una impresora

3.3.2 Diagrama con n impresoras de tickets

En el caso de que se requiera instalar varias impresoras, el servidor se configura con IP estática, los dispositivos de acceso a la aplicación con IP dinámica y las impresoras se configuran con IP estática. La figura 3.41 muestra la topología de red para N impresoras de tickets.

Figura 3.41 Diagrama con *n* impresoras

Capítulo 4
Implementación de casos representativos

4. Implementación de casos representativos

En México existe una amplia variedad de negocios que se clasifican según su número de trabajadores y ventas anuales, en micro, pequeñas, medianas y grandes empresas.

Como se mencionó en el capítulo 1, aproximadamente el 95% de toda la actividad económica dedicada a la venta de comida entra en el sector de micro y pequeñas empresas. La mayoría de estos negocios no cuentan con los recursos suficientes para invertir en sistemas y equipos sofisticados para mejorar su administración, ni poseen el personal capacitado para dar mantenimiento a estos sistemas.

Esto genera una gran problemática para los negocios que apenas comienzan con su operación, ya que corren el riesgo de no sobrevivir durante los primeros 5 años de su vida productiva por falta de una adecuada administración y recursos económicos.

Para dar solución a la problemática planteada en este trabajo, a continuación se describe a nivel general, la estrategia de implementación y desarrollo de los principales componentes del sistema con base al diseño y a la arquitectura planteada en capítulos anteriores.

4.1 Problemática y requisitos particulares del negocio para implementar el sistema

Para propósitos del presente trabajo y con la finalidad de obtener los mejores resultados para su respectivo análisis, se eligió un negocio que posee las siguientes características, cabe aclarar que el sistema está diseñado para ser implementado en micro, pequeñas y medianas empresas, para el caso de grandes empresas se requiere analizar la infraestructura necesaria.

- Flujo de clientes superior a 50 por semana.
- Valor de mercancía en almacén superior a 10 mil pesos.
- Número de trabajadores superior a 5.
- Ventas superiores a 15 mil pesos por mes.

Por motivos de seguridad y privacidad, no se mencionará el nombre ni la ubicación de negocio donde se implementó el sistema, la razón es porque se mostrará información confidencial sobre su operación.

4.1.1 Problemáticas particulares del negocio en donde se implementará el sistema

El negocio que se analizará tiene aproximadamente 8 años de antigüedad, en sus inicios comenzó como un negocio muy simple de comida, donde trabajaban únicamente entre 3 y 4 personas. Al paso del tiempo, el flujo de clientes incrementó considerablemente, y por lo tanto las ventas.

Al incrementarse el flujo de clientes, se tuvo la necesidad de contratar más meseros para poder atender la cantidad de clientes. Sin embargo, esto dio origen a varias problemáticas, todas basadas en el control de los meseros.

Entre una de estas problemáticas se encuentran las confusiones al momento de hacer un pedido, como los pedidos se escribían a mano, muchas veces no se entendía lo que escribían o los mesero hacían pedidos que no existían y no se encontraban dentro de la carta.

También surgieron las perdidas monetarias, al realizar todos los cálculos a mano, no se les cobraba correctamente a los clientes.

Otra problema muy delicado que surgió, fue el robo por parte de los meseros, ya que no entregaban el dinero de todas las cuentas o mesas atendidas durante su turno, al no saber en tiempo real la cantidad exacta de dinero que debía entregar el mesero, se daban con frecuencia este tipo de hechos.

Dadas las problemáticas que presenta el negocio, es factible a que se realice la implementación el sistema de punto de venta, con la finalidad de ayudar a mejorar el control administrativo.

4.2 Implementación de casos representativos.

A continuación, se presenta la estrategia general de implementación de los principales componentes del sistema, así como de la implementación de la arquitectura planteada en capítulos anteriores.

En primera instancia se describe de forma general la instalación y configuración de los componentes principales de software del sistema, el framework Node.js y la base de datos MongoDB. Posteriormente se detalla la implementación de la arquitectura del software y la arquitectura del hardware.

4.2.1 Implementación del servidor de aplicaciones Node.js y la base de datos MongoDB.

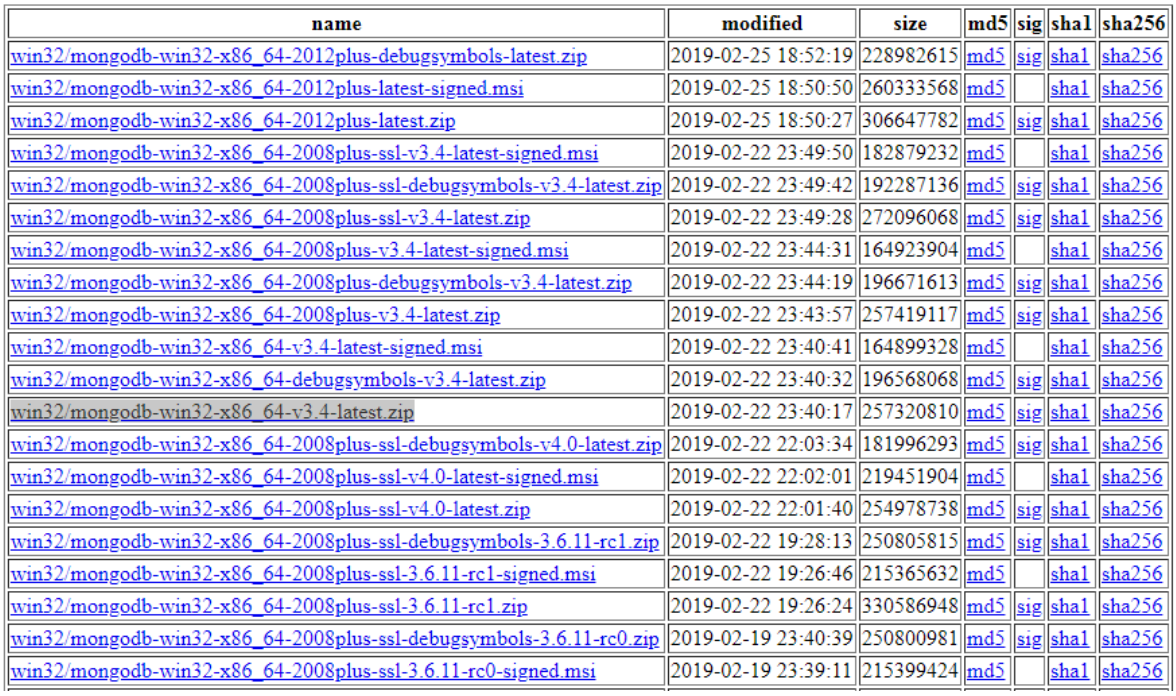
El framework Node.js es multiplataforma, se puede instalar en los principales sistemas operativos que existen actualmente como Windows, MacOs y Linux, lo que resulta muy práctico y portable para instalar en cualquier computadora de escritorio o laptop.

En esta implementación se eligió el sistema operativo Windows, para realizar la instalación de Node.js se eligió la versión v4.8.4[8], con arquitectura a 32 bits, como se muestra en la figura 4.1.

Figura 4.1 Instalador Node.js

En cuanto a la base de datos MongoDB, se hará uso de la versión 3.4[9] a 32 bits como se muestra en la figura 4.2.

[8] https://nodejs.org/en/blog/release/v4.8.4/
[9] https://www.mongodb.org/dl/win32



name	modified	size	md5	sig	sha1	sha256
win32/mongodb-win32-x86_64-2012plus-debugsymbols-latest.zip	2019-02-25 18:52:19	228982615	md5	sig	sha1	sha256
win32/mongodb-win32-x86_64-2012plus-latest-signed.msi	2019-02-25 18:50:50	260333568	md5		sha1	sha256
win32/mongodb-win32-x86_64-2012plus-latest.zip	2019-02-25 18:50:27	306647782	md5	sig	sha1	sha256
win32/mongodb-win32-x86_64-2008plus-ssl-v3.4-latest-signed.msi	2019-02-22 23:49:50	182879232	md5		sha1	sha256
win32/mongodb-win32-x86_64-2008plus-ssl-debugsymbols-v3.4-latest.zip	2019-02-22 23:49:42	192287136	md5	sig	sha1	sha256
win32/mongodb-win32-x86_64-2008plus-ssl-v3.4-latest.zip	2019-02-22 23:49:28	272096068	md5	sig	sha1	sha256
win32/mongodb-win32-x86_64-2008plus-v3.4-latest-signed.msi	2019-02-22 23:44:31	164923904	md5		sha1	sha256
win32/mongodb-win32-x86_64-2008plus-debugsymbols-v3.4-latest.zip	2019-02-22 23:44:19	196671613	md5	sig	sha1	sha256
win32/mongodb-win32-x86_64-2008plus-v3.4-latest.zip	2019-02-22 23:43:57	257419117	md5	sig	sha1	sha256
win32/mongodb-win32-x86_64-v3.4-latest-signed.msi	2019-02-22 23:40:41	164899328	md5		sha1	sha256
win32/mongodb-win32-x86_64-debugsymbols-v3.4-latest.zip	2019-02-22 23:40:32	196568068	md5	sig	sha1	sha256
win32/mongodb-win32-x86_64-v3.4-latest.zip	2019-02-22 23:40:17	257320810	md5	sig	sha1	sha256
win32/mongodb-win32-x86_64-2008plus-ssl-debugsymbols-v4.0-latest.zip	2019-02-22 22:03:34	181996293	md5	sig	sha1	sha256
win32/mongodb-win32-x86_64-2008plus-ssl-v4.0-latest-signed.msi	2019-02-22 22:02:01	219451904	md5		sha1	sha256
win32/mongodb-win32-x86_64-2008plus-ssl-v4.0-latest.zip	2019-02-22 22:01:40	254978738	md5	sig	sha1	sha256
win32/mongodb-win32-x86_64-2008plus-ssl-debugsymbols-3.6.11-rc1.zip	2019-02-22 19:28:13	250805815	md5	sig	sha1	sha256
win32/mongodb-win32-x86_64-2008plus-ssl-3.6.11-rc1-signed.msi	2019-02-22 19:26:46	215365632	md5		sha1	sha256
win32/mongodb-win32-x86_64-2008plus-ssl-3.6.11-rc1.zip	2019-02-22 19:26:24	330586948	md5	sig	sha1	sha256
win32/mongodb-win32-x86_64-2008plus-ssl-debugsymbols-3.6.11-rc0.zip	2019-02-19 23:40:39	250800981	md5	sig	sha1	sha256
win32/mongodb-win32-x86_64-2008plus-ssl-3.6.11-rc0-signed.msi	2019-02-19 23:39:11	215399424	md5		sha1	sha256

Figura 4.2 Descarga instalador MongoDB

La instalación se realiza de forma local, no se instalará en forma de clúster. Para inicializar el servidor de MongoDB, se ejecuta `mongod.exe` que se encuentra dentro del directorio en el que se instaló MongoDB.

Con esto ya se tiene el servidor MongoDB esperando a recibir conexiones desde el puerto `27017`.

4.2.2 Implementación de la arquitectura de software

Como se mencionó en el capítulo anterior, para el desarrollo del sistema de punto de venta se eligió el modelo multicapa, que consiste en separar los módulos de base de datos, presentación y de negocio. Dentro de la capa de presentación se emplea el Modelo-Vista-Controlador (MVC).

Igualmente se mencionó acerca de los componentes que hacen referencia a la parte de la presentación de datos del lado del cliente (Front-End) y los que pertenecen a los del lado del servidor (Back-End), que son componentes que se encargan como por ejemplo de la persistencia de los datos.

A continuación, en la figura 4.3 se presenta el diagrama de capas, implementado en el sistema de punto de venta.

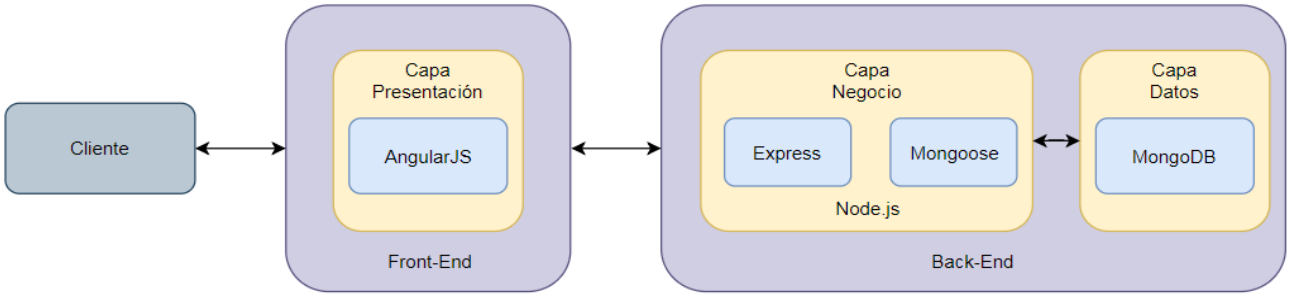


Figura 4.3 Diagrama de capas del sistema de punto de venta

Como se observa en la figura 4.3, se integran varios componentes y tecnologías dentro del sistema de punto de venta. Para la parte de la presentación de datos, se emplea el framework JavaScript AngularJS, en conjunto con HTML y Bootstrap. Estas herramientas permiten la construcción de las vistas con las que el usuario interactúa, ya sea para realizar solicitudes al sistema o para visualizar los datos resultantes de una operación.

Para ejemplificar la capa de presentación, a continuación en la figura 4.4, se muestra la interfaz gráfica para el alta de productos nuevos en el sistema. Aquí se hace uso de formularios para ingresar información y de validación de datos por medio de Bootstrap.

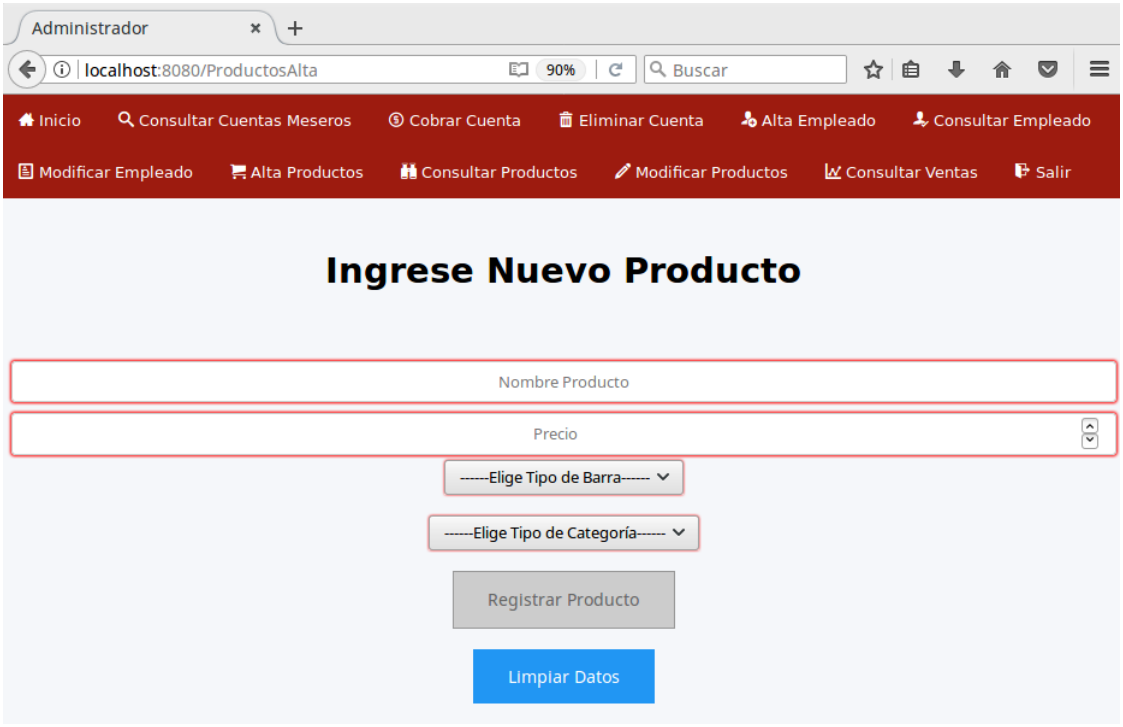


Figura 4.4 Interfaz gráfica alta productos

Para ingresar datos en el formulario, Bootstrap realiza las validaciones necesarias, si los datos son válidos, se habilita el botón para registrar el producto. En la figura 4.5 se muestra un ejemplo de registro de un producto.

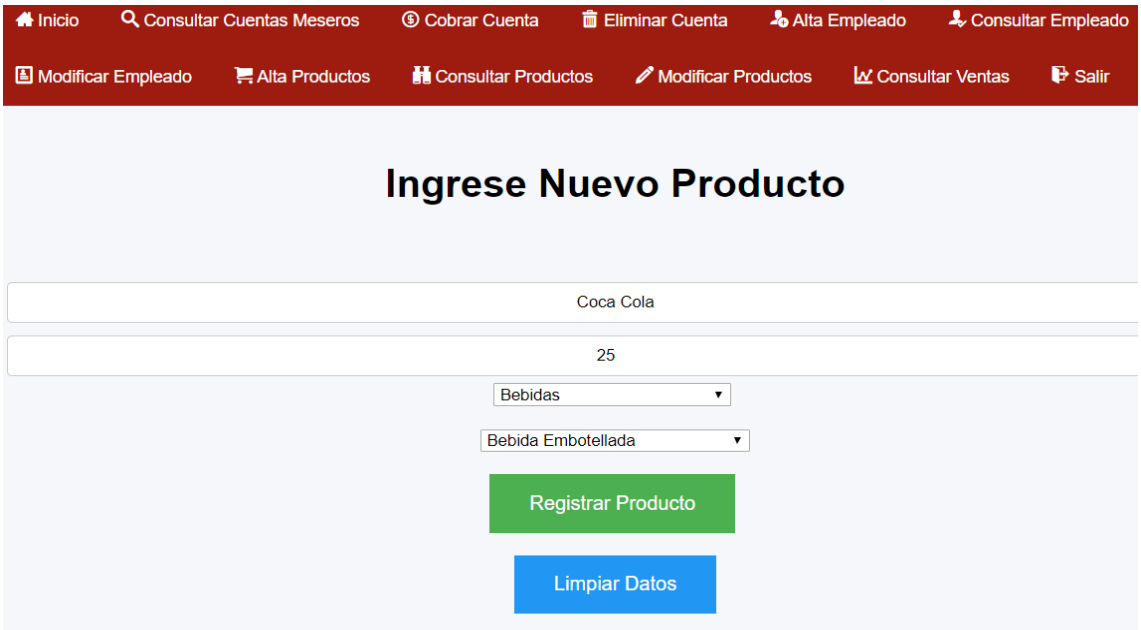


Figura 4.5 Producto ejemplo alta productos

En el código HTML de la figura 4.6, se indica cómo se integra la herramienta de AngularJS para construir páginas web.

```
<html ng-app="MainApp">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <script src="./Librerias/angular-1.2.32/angular.min.js"></script>
    <link rel="stylesheet" href="fonts.css">
    <script src="core.js"></script>
    <link rel="stylesheet" href="styleAdmin.css">
    <title>Administrador</title>
  </head>
  <body class="body" ng-controller="mainController">
    . . . .
  </body>
</html>
```

Figura 4.6 Código fuente implementación AngularJS.

Una vez que el usuario ha solicitado la acción de registrar el producto mediante la interfaz gráfica, se invocan algunos elementos y servicios de la capa de negocio, entre los que destaca, se encuentran los llamados `SCOPE` que son elementos que se encargan de transportar datos entre la capa de presentación y la capa de negocio. En el código de la figura 4.7, se muestra como se definen estos elementos.

```
angular.module('MainApp', [])
  function mainController($scope, $http) {
    $scope.newPersona = {};
    $scope.personas = {};
    $scope.newProducto = {};
    $scope.productos = {};
    $scope.newInfo_cuenta = {};
    $scope.info_cuenta = {};
    $scope.newPedido = {};
    $scope.pedidos = {};
    $scope.selected = false;
    $scope.date = new Date();
    $scope.categoria;
    $scope.datosHeader = {};
    ...
```

Figura 4.7 Código fuente definición Scope

Para poder dar de alta el producto o crear el objeto en la base de datos es necesario invocar un servicio encargado de construir los métodos del API REST (Representational State Transfer - Transferencia de Estado Representacional). Estos servicios se encuentran dentro del módulo de Express en la capa de negocio y define los siguientes métodos:

- GET - Consultar objetos
- POST - Crear objetos
- PUT - Actualizar objetos
- DELETE - Eliminar objetos

Para el caso del alta de productos, se utiliza el método POST. El objetivo es construir la ruta HTTP. Aquí se indica el tipo de método del API REST y en caso de ser necesario también se le especifican los parámetros, la forma de cómo se declara la construcción de la ruta, se muestra en el código de la figura 4.8

```
$scope.registrarProducto = function() {
  if (confirm("Estás seguro en dar de Alta el Producto?")) {
    $http.post('/api/productos', $scope.newProducto)
      .success(function(data) {$scope.newProducto = {};
    $scope.productos = data;}).error(function(data) {
    console.log('Error: ' + data);
    });
  }
}; ...
```

Figura 4.8 Código fuente API REST

Existe un servicio dedicado a capturar las rutas llamado ROUTE. Igualmente se encuentra dentro del módulo de Express en la capa de negocio. Este servicio se encarga de redirigir la solicitud al método que realiza la acción sobre la base de datos. En la figura 4.9 se muestra como se implementa la captura de la ruta, las validaciones de datos y se redirige la solicitud al método encargado de realizar la acción sobre la base de datos.

```
//Captura la ruta de la solicitud HTTP
app.post('/api/productos', isLoggedInAdm, controllerProductos.setProductos);
```

Figura 4.9 Código fuente ROUTE

Cuando se captura la ruta y el módulo `ROUTE` de `Express` ya se conoce el método a invocar para realizar la acción, consecuentemente se hace uso del módulo de `Mongoose`. Este módulo se encarga de realizar la conexión con la base de datos de MongoDB, también conocido como driver. Aquí se indican las instrucciones que se deben de realizar sobre la base de datos, es el encargado de hacer la traducción de las instrucciones que se realizan desde el sistema a instrucciones que entienda la base de datos, en este caso, crear un objeto nuevo. En la figura 4.10 se muestra la implementación del módulo de `Mongoose`, y el tipo de operación que se realiza en la base de datos.

```
//Funcion crea producto nuevo
exports.setProductos = function(req, res) {
  Producto.create({
  producto: req.body.producto,
  precio: req.body.precio,
  categoria: req.body.categoria,
  tipo_barra: req.body.tipo_barra
},
...
```

Figura 4.10 Código fuente Mongoose

En esta misma sección de código, también se indican las acciones a realizar en caso de que la operación finalice correctamente o en error, también es conocido como llamadas de retorno o callback. En el ejemplo de código de la figura 4.11, se muestra como se implementan estos callbacks.

```
//Función Callback
...
function(err, productos) {
  if (err)
    res.send(err);
  Producto.find(function(err, productos) {
  if (err)
    res.send(err);
  res.json(productos);
});
```

Figura 4.11 Función callback crea producto

Finalmente y para validar que los datos sean insertado correctamente en la base de datos, se realiza una consulta en MongoDB, como se muestra en la figura 4.12.

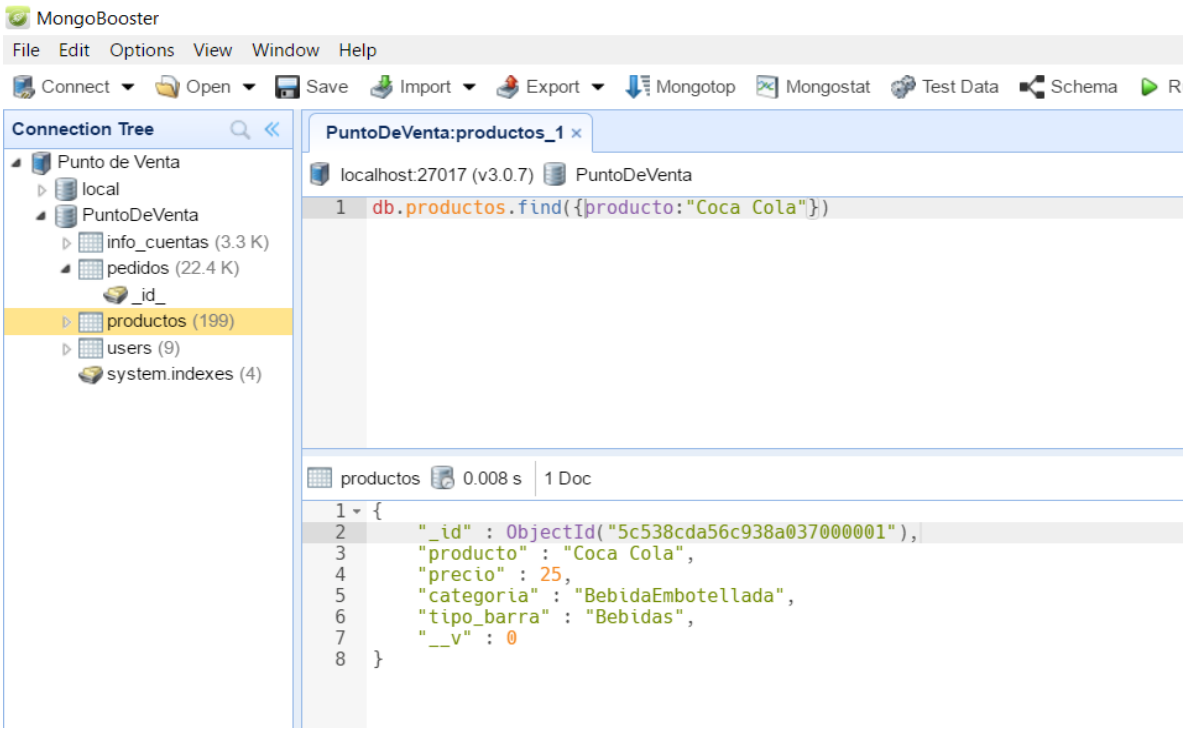


Figura 4.12 Consulta datos en MongoDB

4.2.3 Implementación de manejo de eventos

Para el manejo de eventos, Node.js implementa un mecanismo que permite procesar las solicitudes que se envían al sistema. El desarrollador no tiene la necesidad de escribir código para el manejo de estos eventos ya que estos se manejan internamente. En Node.js, solo basta con programar las solicitudes y las llamadas de retorno (callbacks). En la figura 4.13 se muestra la estructura general del mecanismo de procesamiento de peticiones.

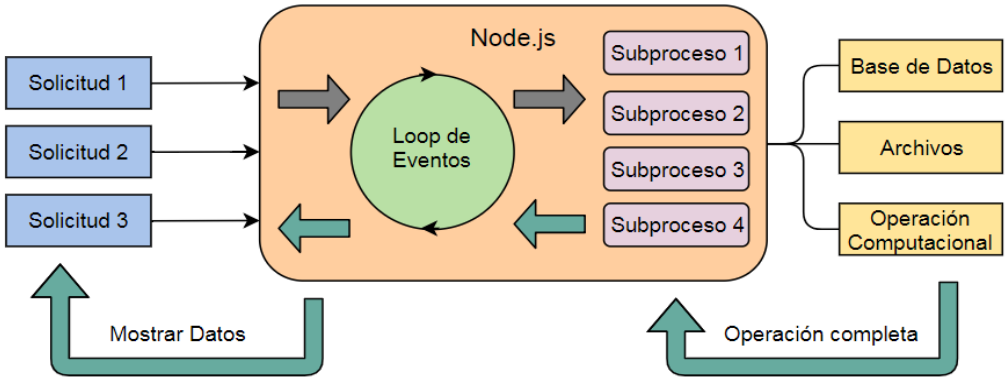


Figura 4.13 Diagrama de eventos Node.js

Una vez que Node.js captura una solicitud, esta pasa por el loop de eventos y se genera un subproceso, que es controlado por la librería `Libuv`, la cual se auxilia del sistema operativo o la base de datos, para terminar de procesar la operación solicitada. En el código de la figura 4.14, se muestra un ejemplo de una solicitud sobre una operación de actualización sobre la base de datos.

```
Producto.update({
  _id: req.params.productos_id}, {
    $set: {
      producto: req.body.producto,
      precio: req.body.precio,
      categoria: req.body.categoria,
      tipo_barra: req.body.tipo_barra
    }
}...
```

Figura 4.14 Código de solicitudes

Esta solicitud es capturada por el loop de eventos Node.js y ejecuta la acción sobre la base de datos. Al término de su ejecución, se le notifica al loop de eventos sobre el resultado y este inmediatamente ejecuta las acciones del callback correspondiente. En el código de la figura 4.15 se observan las acciones que se realizan después de que finaliza la operación.

```
function(err, productos) {
  if (err)
    res.send(err);
  else{
    Producto.find(function(err, productos) {
    if (err)
      res.send(err);
    res.json(productos);
  });
}
```

Figura 4.15 Código de función callback

La respuesta de la función callback se propaga a la capa de presentación donde se indica que datos mostrar después de la acción completada. En la figura 4.16 se muestra como se propaga la función callback a los `SCOPE` que se explicaron anteriormente.

```
$http.get('/api/ProductosActualizarBD/' + datosConsulta + '/' + optionSearch)
  .success(function(data) {
    $scope.productos = data;
    $scope.newProducto = {};
  })
  .error(function(data) {
    console.log('Error: ' + data);
  });
```

Figura 4.16 Código de función callback muestra datos

4.3 Implementación de la arquitectura del hardware

A continuación en la tabla 4.17, se describen las características del hardware que se utilizará.

Dispositivo	Características
Computadora Central	Sistema Operativo: Windows 7, 32 bits Memoria Ram: 2 GB Disco Duro: 500 GB
Router	Asus, RT-N300, 300Mbps, dos antenas externas de 5dBi
Impresora de Tickets	Impresora térmica Bixolon F-310
Tablets o teléfono celular	Sistema Operativo: Versión Android superior a 4.2 Memoria Ram: 1 GB Memoria de Almacenamiento: superior a 2 GB

Tabla 4.17 Características de los dispositivos

La comunicación entre los dispositivos de acceso a la aplicación (tablets, teléfonos celulares) y el servidor (computadora central) se realiza por medio de una red de área local inalámbrica (WLAN), para este caso el sistema funciona con una impresora de tickets.

Entre las características de conexión de los dispositivos de red, están: el servidor de aplicaciones tiene IP estática, los dispositivos de acceso a la aplicación tienen IP dinámica y la impresora se configura por medio de cable USB en el servidor, como se muestra en la figura 4.18.

Figura 4.18 Topología de red

Capítulo 5
Pruebas y análisis de resultados

5. Pruebas y análisis de resultados

El propósito principal de realizar pruebas sobre la aplicación es mejorar la calidad del sistema, detectar alguna falla o evento inesperado, antes de ser liberado a producción.

En esta sección se describe algunos tipos de pruebas:

- Pruebas unitarias - son pruebas de bajo nivel, se analiza el resultado de las funciones y métodos de los componentes del sistema.
- Pruebas funcionales - en este tipo de pruebas se analizan las reglas del negocio y la validación de datos.
- Pruebas integrales o de integración - se verifica el correcto funcionamiento en conjunto de todos los módulos del sistema.
- Pruebas de rendimiento - validar el tiempo de respuesta del sistema bajo distintas condiciones de carga de trabajo.
- Pruebas de humo - son pruebas rápidas, se valida que los principales módulos del sistema funcionen de acuerdo a lo previsto.

Otro aspecto que se debe considerar al momento de realizar pruebas, es lo conveniente que resulta automatizarlas, la capacidad humana de realizar pruebas manuales es limitada, en cambio con la automatización de estas tareas repetitivas, puede ayudar a simplificar y agilizar esta labor.

Este tipo de pruebas automatizadas comúnmente son implementadas cuando ya se tiene definido un rango de valores que se desean validar dentro de la aplicación.

Para propósitos de este trabajo, únicamente se analizan dos tipos de pruebas manuales, unitarias y funcionales.

5.1 Pruebas unitarias

Para realizar este tipo de pruebas en Node.js, existen varias herramientas, entre ellas se encuentran los frameworks Mocha y Chai, cuya función es notificar el resultado de la prueba solicitada.

Este tipo de frameworks poseen características para realizar varias validaciones en las pruebas, como por ejemplo que tipo de respuesta se espera o cual debería de ser.

Para este tipo de pruebas unitarias se programan dos módulos principalmente:

- `describe` - Aquí se indica la descripción sobre qué consiste la prueba.
- `it` - Es la implementación de las validaciones del módulo que se está probando, igualmente aquí se escriben las reglas de validación.

Para este caso se realizan pruebas sobre los métodos `GET`, `POST`, `PUT` y `DELETE`, se valida si la respuesta `HTTP` es satisfactoria (`status 200`) y que la operación solicitada se vea reflejada en la base de datos.

En el método `GET` se analiza la extracción de información de la base de datos. Como se muestra en la figura 5.1 se extraen todos los productos de la base de datos.

```
describe('# GET - Productos', function() {
  it('Obtiene productos /api/productos GET', function(done) {
  chai.request(server)
    .get('/api/productos')
    .end(function(err, res){
      res.should.have.status(200);
      done();
    });
  });
});
```

Figura 5.1 Código fuente prueba unitaria `GET`

En la imagen de la figura 5.2 se muestra la ejecución de la prueba del método `GET`.

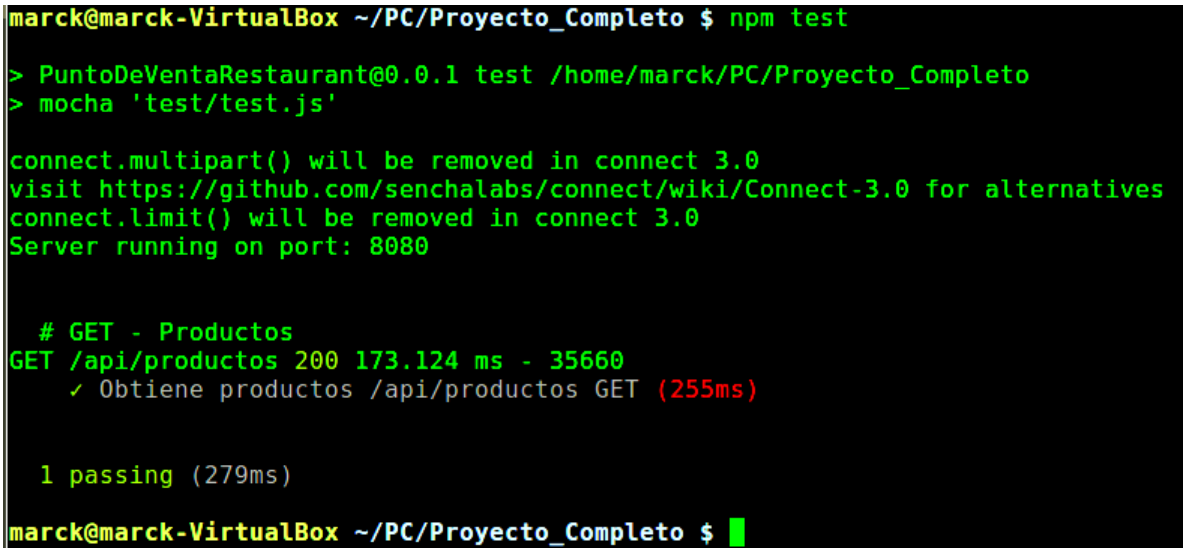


Figura 5.2 Ejecución prueba `GET`

En el ejemplo del código fuente de la figura 5.3 se muestra la prueba del método `POST`, para crear un nuevo producto en la base de datos.

```
describe('# POST - Productos', function() {
  it('Agrega Producto /api/productos POST', function(done) {
  chai.request(server)
    .post('/api/productos')
    .send({
    'producto' : 'Prueba1',
    'precio' : 0,
    'categoria' : 'CategoriaPrueba',
    'tipo_barra' : 'TipoBarraPrueba'
    })
    .end(function(err, res){
      res.should.have.status(200);
      done();
    });
  });
});
```

Figura 5.3 Código fuente prueba unitaria `POST`

En la imagen de la figura 5.4 se muestra la ejecución de la prueba del método `POST`.

```
marck@marck-VirtualBox ~/PC/Proyecto_Completo $ npm test

> PuntoDeVentaRestaurant@0.0.1 test /home/marck/PC/Proyecto_Completo
> mocha 'test/test.js'

connect.multipart() will be removed in connect 3.0
visit https://github.com/senchalabs/connect/wiki/Connect-3.0 for alternatives
connect.limit() will be removed in connect 3.0
Server running on port: 8080


  # POST - Productos
POST /api/productos 200 198.142 ms - 35838
    ✓ Agrega Producto /api/productos POST (282ms)


  1 passing (307ms)

marck@marck-VirtualBox ~/PC/Proyecto_Completo $
```

Figura 5.4 Ejecución prueba `POST`

En la figura 5.5 se muestra el producto de prueba creado en la base de datos.

Figura 5.5 Consulta resultado base de datos prueba `POST`

En el ejemplo del código fuente de la figura 5.6 se muestra la prueba del método `PUT`, para actualizar un producto en la base de datos.

```
describe('# PUT - Productos', function() {
  it('Actualiza Producto /api/productos/<id> PUT', function(done) {
      chai.request(server)
        .put('/api/productos/'+5c897666cc75a0ac04000001)
        .send({
        'producto' : 'Prueba2',
        'precio' : 0,
        'categoria' : 'CategoriaPrueba',
        'tipo_barra' : 'TipoBarraPrueba'
        })
        .end(function(error, response){
          response.should.have.status(200);
          done();
      });
  });
});
```

Figura 5.6 Código fuente prueba unitaria `PUT`

En la imagen de la figura 5.7 se muestra la ejecución de la prueba del método `PUT`.

```
marck@marck-VirtualBox ~/PC/Proyecto_Completo $ npm test

> PuntoDeVentaRestaurant@0.0.1 test /home/marck/PC/Proyecto_Completo
> mocha 'test/test.js'

connect.multipart() will be removed in connect 3.0
visit https://github.com/senchalabs/connect/wiki/Connect-3.0 for alternatives
connect.limit() will be removed in connect 3.0
Server running on port: 8080


  # PUT - Productos
PUT /api/productos/5c897666cc75a0ac04000001 200 188.671 ms - 35838
    ✓ Actualiza Producto /api/productos/<id> PUT (276ms)


  1 passing (316ms)

marck@marck-VirtualBox ~/PC/Proyecto_Completo $
```

Figura 5.7 Ejecución prueba `PUT`

En la figura 5.8 se muestra el producto actualizado en la base de datos.

Figura 5.8 Consulta resultado base de datos prueba `PUT`

En el ejemplo del código fuente de la figura 5.9 se muestra la prueba del método `DELETE`, para borrar un producto en la base de datos.

```
describe('# DELETE - Productos', function() {
  it('Elimina Producto /api/productos/<id> DELETE', function(done) {
      chai.request(server)
        .delete('/api/productos/'+'5c897666cc75a0ac04000001')
        .end(function(error, response){
          response.should.have.status(200);
          done();
      });
  });
});
```

Figura 5.9 Código fuente prueba unitaria `DELETE`

En la imagen de la figura 5.10 se muestra la ejecución de la prueba del método `DELETE`.

```
marck@marck-VirtualBox ~/PC/Proyecto_Completo $ npm test

> PuntoDeVentaRestaurant@0.0.1 test /home/marck/PC/Proyecto_Completo
> mocha 'test/test.js'

connect.multipart() will be removed in connect 3.0
visit https://github.com/senchalabs/connect/wiki/Connect-3.0 for alternatives
connect.limit() will be removed in connect 3.0
Server running on port: 8080


  # DELETE - Productos
DELETE /api/productos/5c897666cc75a0ac04000001 200 169.396 ms - 35660
    ✓ Elimina Producto /api/productos/<id> DELETE (252ms)


  1 passing (290ms)

marck@marck-VirtualBox ~/PC/Proyecto_Completo $
```

Figura 5.10 Ejecución prueba `DELETE`

En la figura 5.11 se muestra el producto borrado de la base de datos.

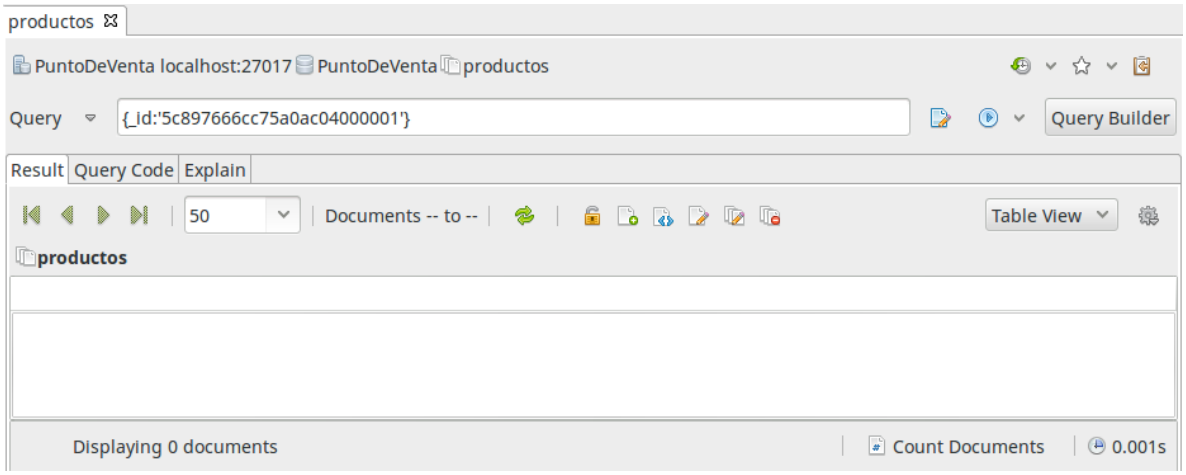


Figura 5.11 Consulta resultado base de datos prueba `DELETE`

5.2 Pruebas funcionales

Para las pruebas funcionales del sistema, se consideran los siguientes factores para recolectar una suficiente cantidad de datos para su respectivo análisis.

- Dispositivos de acceso a la aplicación:
  - 1 computadora de escritorio como servidor de aplicaciones
  - 2 tablets con Android
  - 4 teléfonos celulares con Android
- Tiempo para la recolección de datos: 6 meses
- Carga inicial: catálogo de productos y usuarios.
- Número de meseros: 5

Entre las pruebas funcionales se considera analizar los módulos de autentificación, altas, bajas y cambios en la información, en cada uno de los perfiles del sistema.

5.2.1 Autentificación

Para facilitar el acceso a la aplicación, los usuarios solo ingresan con su password, el sistema tiene la capacidad de identificar el tipo de perfil al que pertenece cada usuario,

si es mesero, cajero o administrador, así como se muestran en la figura 5.12, 5.13, 5.14 y 5.15, la interfaz gráfica de cada perfil.

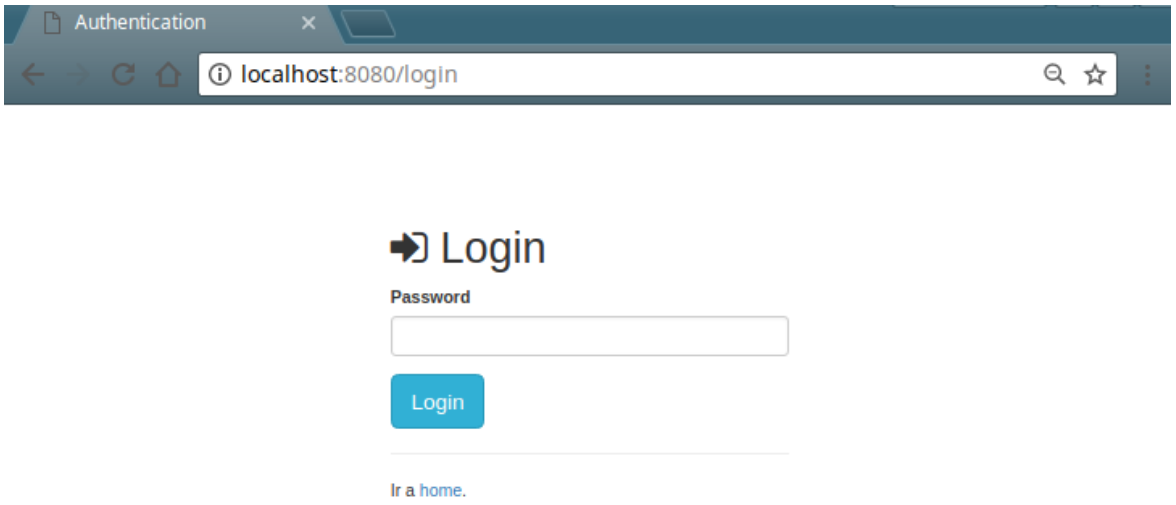


Figura 5.12 Interfaz gráfica de autentificación

Figura 5.13 Interfaz gráfica del mesero

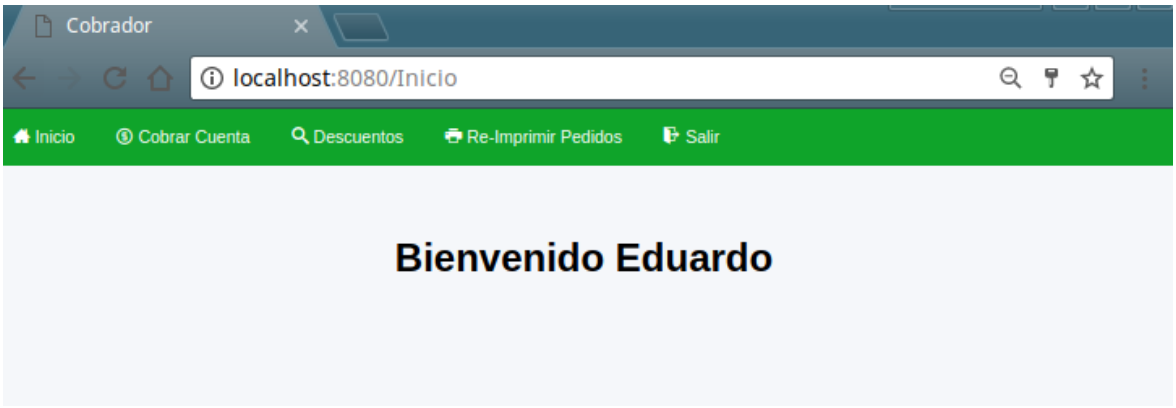


Figura 5.14 Interfaz gráfica del cajero

Figura 5.15 Interfaz gráfica del administrador

5.2.2 Pruebas perfil mesero

En el perfil del mesero se probarán los siguientes puntos:

- Crear una cuenta nueva

Para crear una cuenta nueva se accede al menú de *Nuevo Pedido*, como se muestra en la figura 5.16

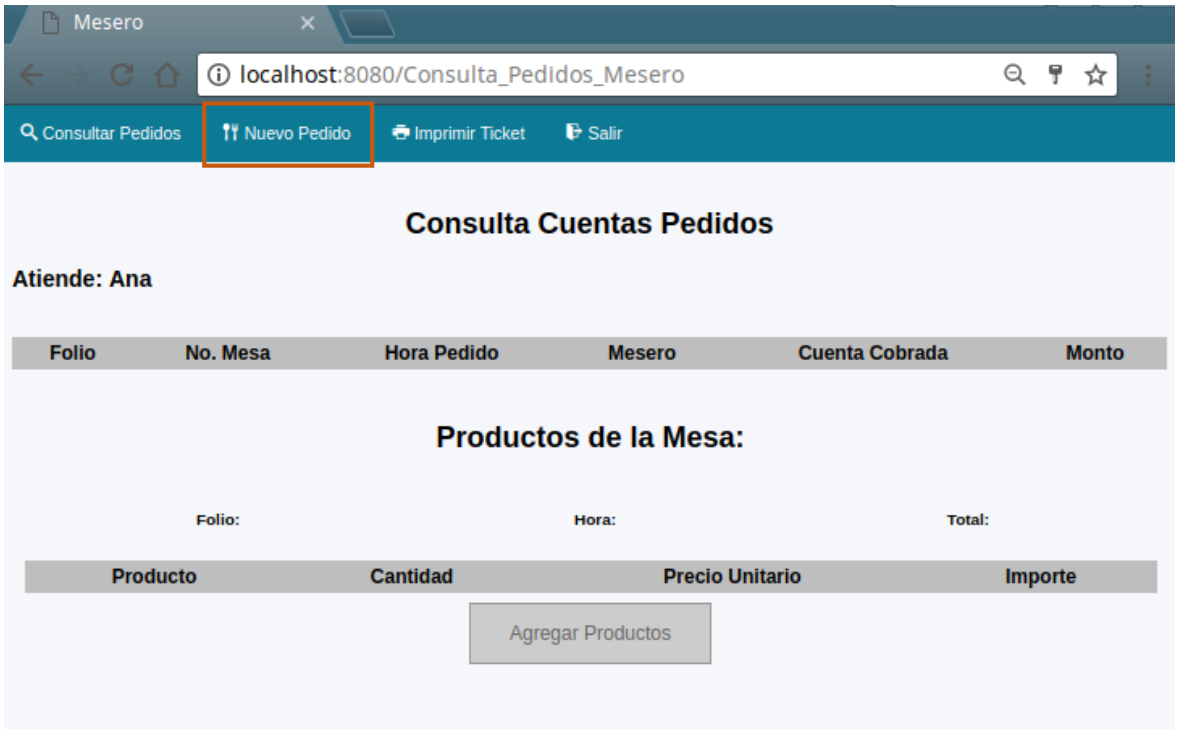


Figura 5.16 Nuevo Pedido

Al Seleccionar *Nuevo Pedido* se mostrará la interfaz para colocar el número de clientes y el número de mesa, como se muestra en la figura 5.17

Figura 5.17 Nuevo Pedido, captura número de mesa y clientes

Inmediatamente se abre otra interfaz gráfica para comenzar a capturar los pedidos, como se muestra en la figura 5.18

Figura 5.18 Nuevo pedido, captura pedidos

La interfaz para capturar pedidos está compuesta por las siguientes secciones: comanda (lista de pedidos agregados a la cuenta), captura de cantidad y observaciones de los pedidos, búsqueda y selección de producto, como se muestra en la figura 5.19.

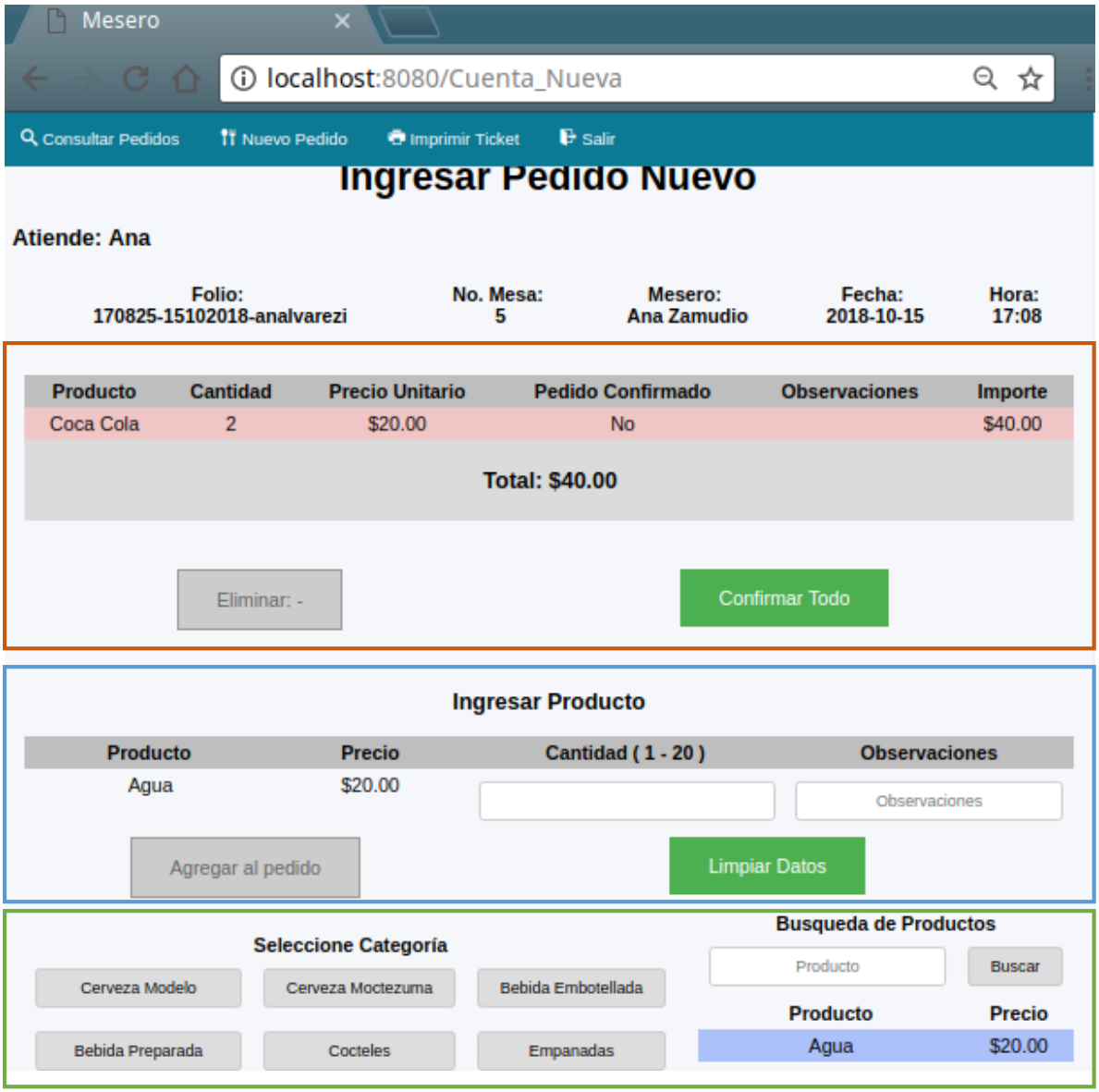


Figura 5.19 Secciones captura pedidos

En la sección de comanda es donde se muestran los productos que se van agregando a la cuenta, en la sección de capturar pedido, es donde se indica la cantidad del producto y las observaciones y finalmente en la sección de búsqueda y selección de productos, es donde se muestra todos los productos que pueden ser seleccionados para ingresarlos como pedidos.

Al oprimir el botón de *Confirmar Todo*, el ticket del pedido se imprime en su correspondiente estación de servicio, como se muestra en la figura 5.20

```
### BEBIDAS - No.Ticket - 1 ###

 Cantidad - Productos - Observaciones

 2 - Coca Cola -  N/A

Nombre Mesero: Ana Zamudio
Folio: 170825-15102018-analvarezi
Hora Pedido: 17_37_49
No. Mesa: 5
```

Figura 5.20 Ticket de pedido

- Agregar productos a una cuenta existente

Para agregar pedidos a una cuenta existen se tiene que acceder al menú y dar click en *Consultar Pedidos*, como se muestra en la figura 5.21

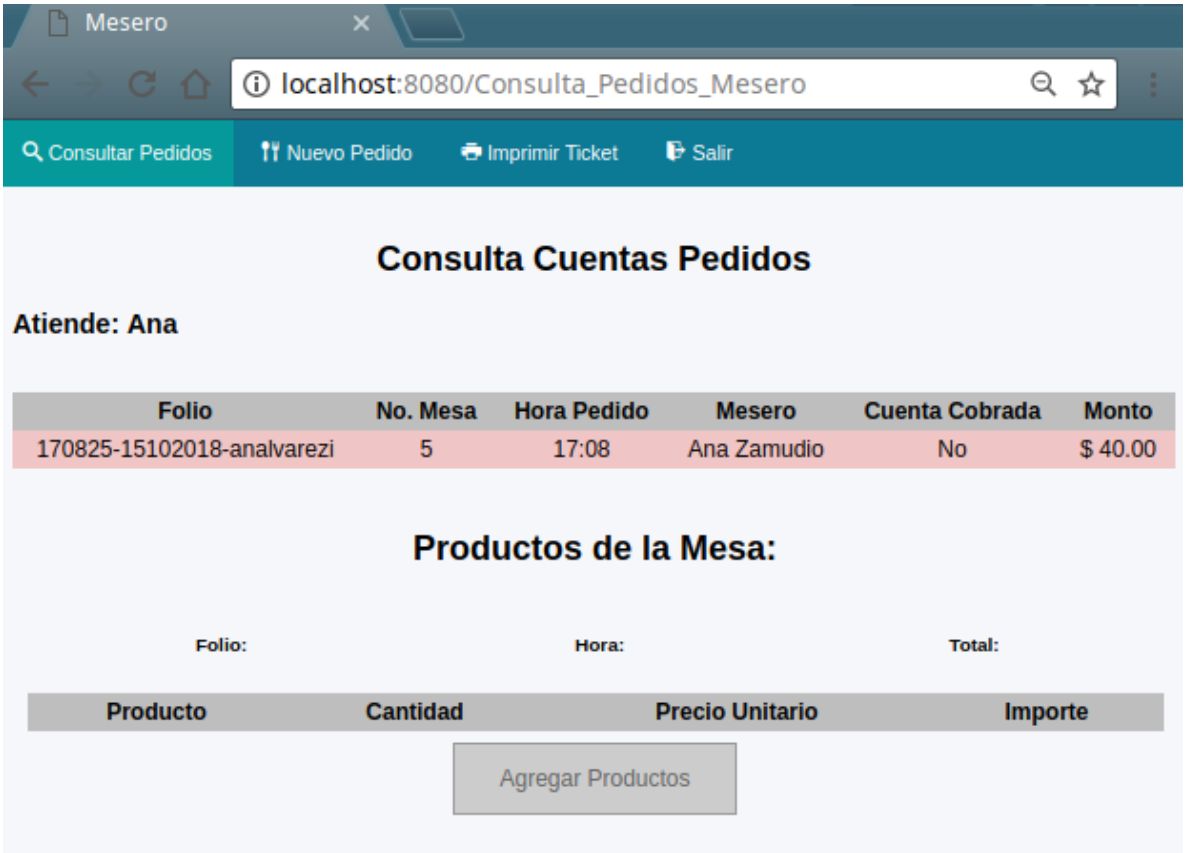


Figura 5.21 Consultar Pedidos

Elegimos la cuenta a la que se le quiere agregar más productos y en la parte inferior aparecerán los productos que ya contiene la cuenta, como se ilustra en la figura 5.22

Figura 5.22 Elegir cuenta para agregar más productos

Al hacer click en *Agregar Productos* volverá aparecer la interfaz gráfica para agregar productos, como se muestra en la figura 5.23

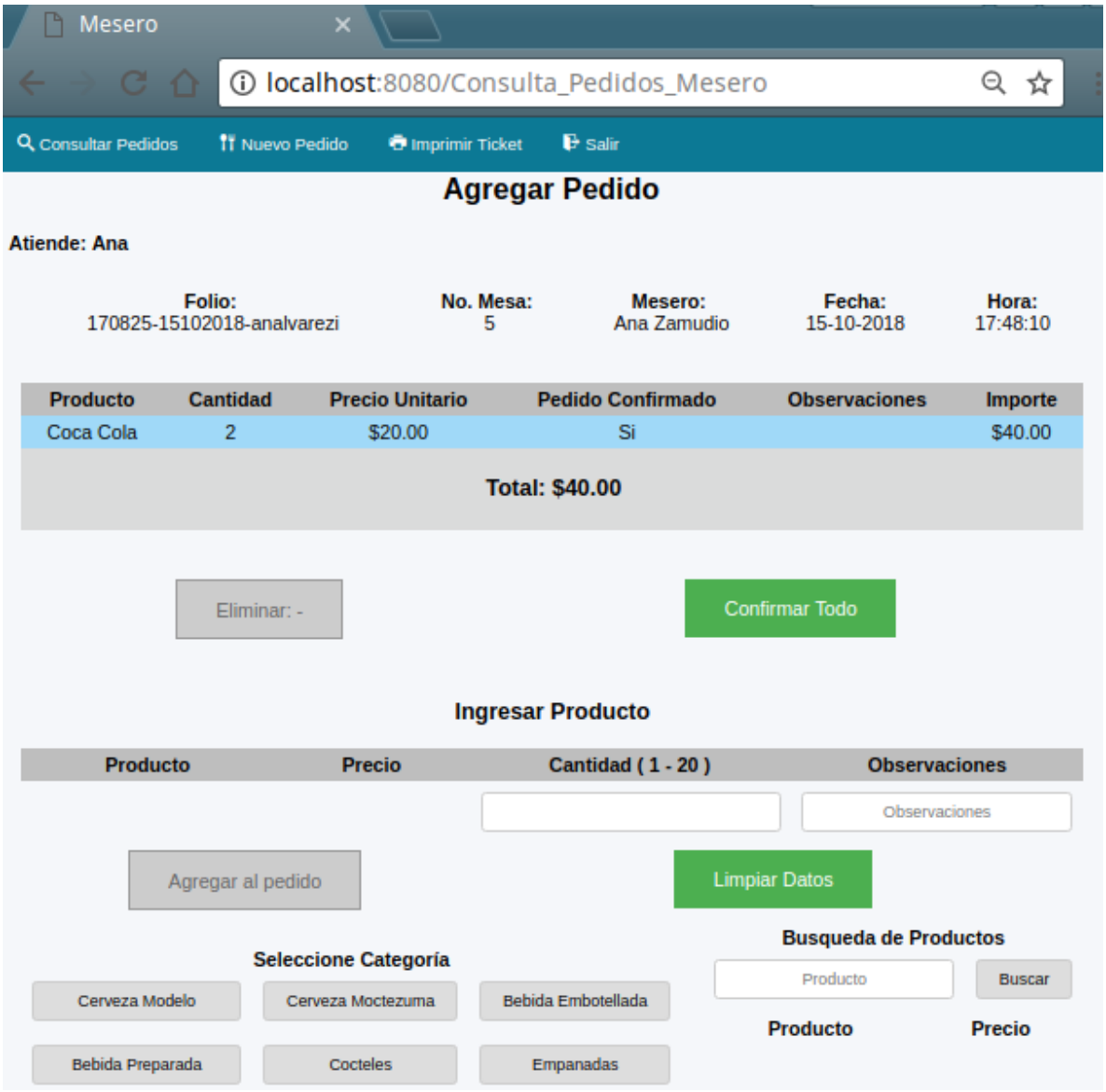


Figura 5.23 Agregar más productos a una cuenta existente

- Imprimir ticket del total de la cuenta

Para imprimir el ticket del total de la cuenta en el menú se ingresa a la sección de *Imprimir Ticket*, como se muestra en la figura 5.24

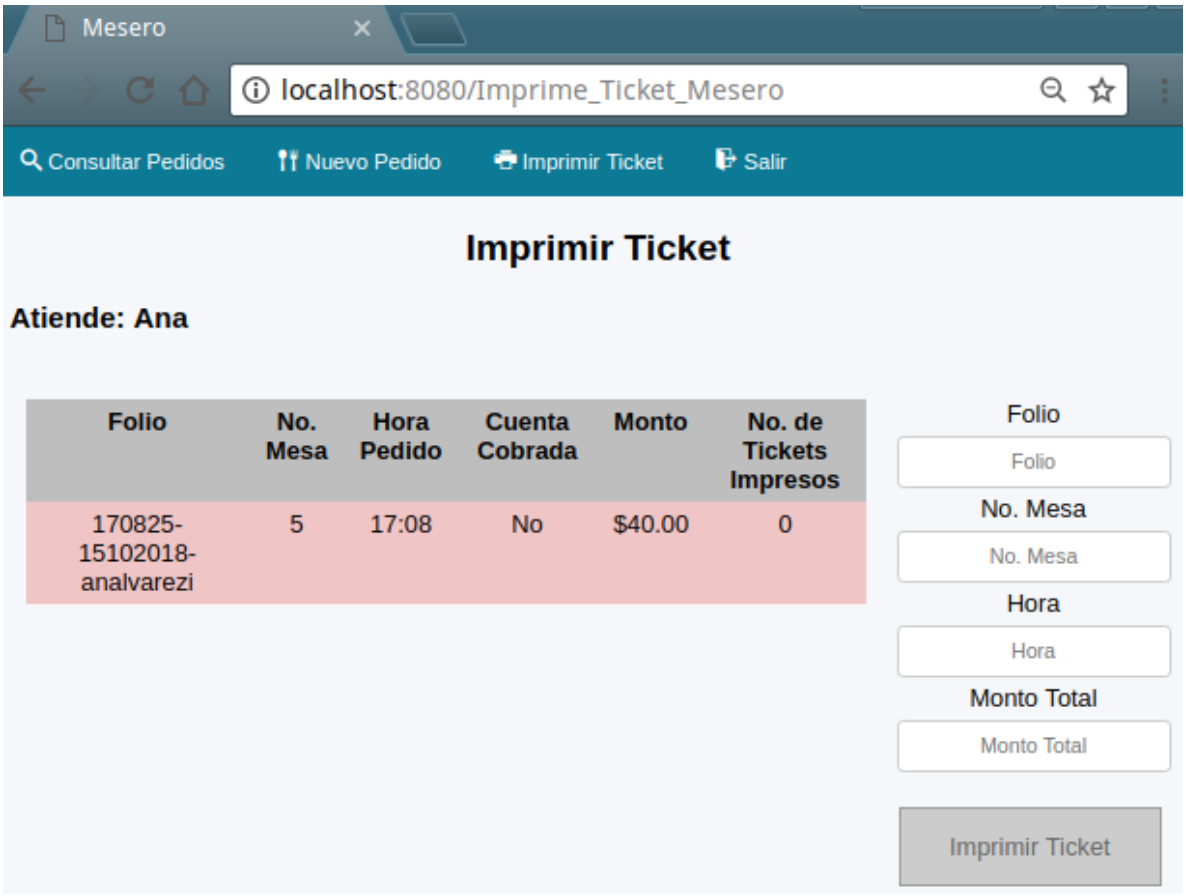


Figura 5.24 Imprimir ticket del total de la cuenta

Se selecciona la cuenta que se va imprimir como se muestra en la figura 5.25

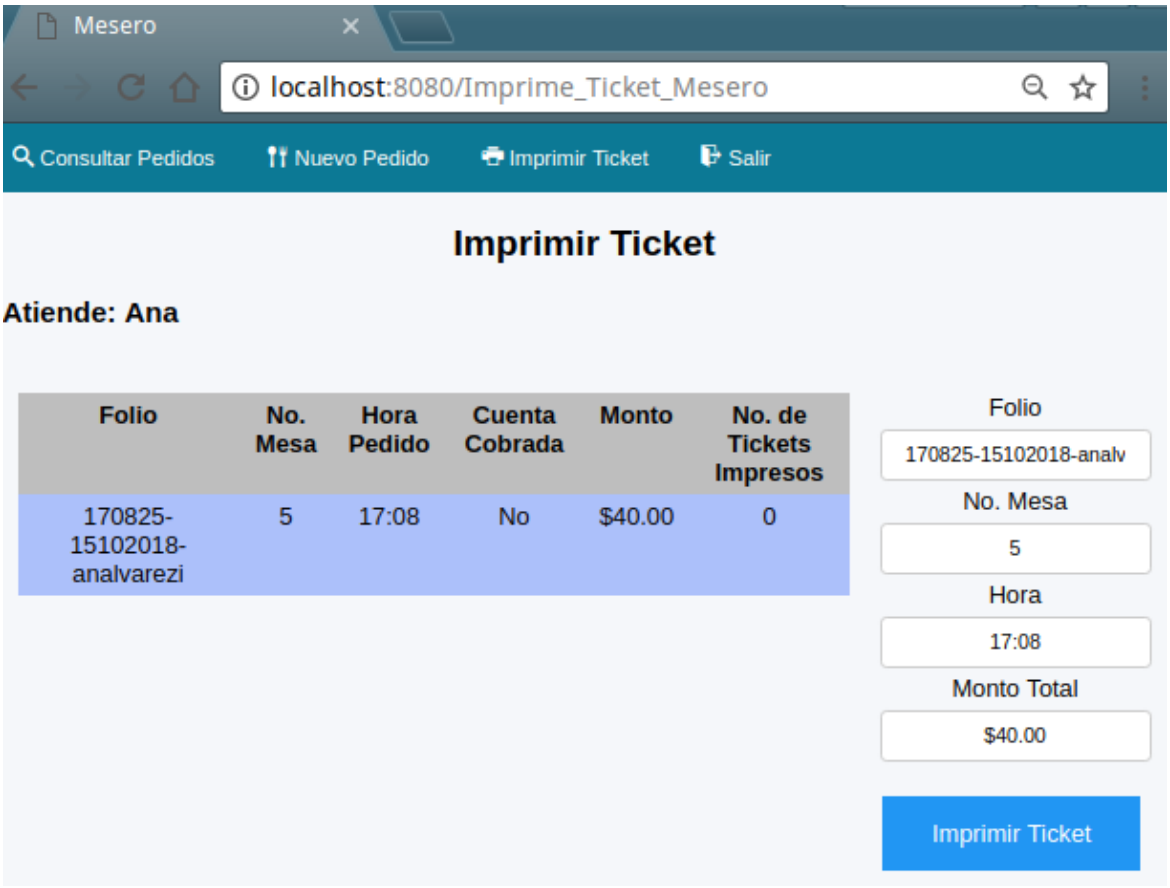


Figura 5.25 Selección de la cuenta para imprimir

Cuando se ha seleccionado la cuenta, se oprime el botón *Imprimir Ticket*, e inmediatamente se realiza la impresión del ticket, como se muestra en la figura 5.26

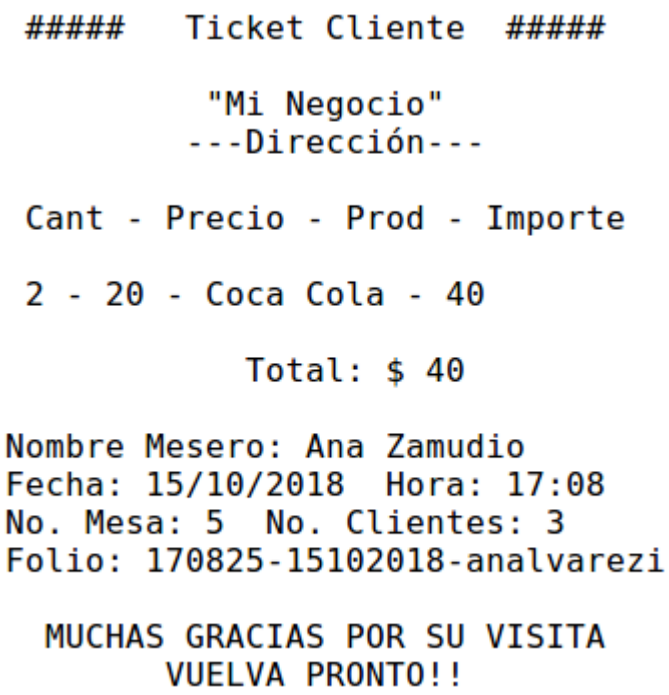


Figura 5.26 Ticket del total de la cuenta

5.2.3 Pruebas perfil cajero

En el perfil del cajero se probaran las siguientes funcionalidades:

- Cobrar una cuenta y cerrar cuenta.

Para cobrar una cuenta, en el menú se selecciona *Cobrar Cuenta* con el perfil de cajero y se elige la cuenta que se desea cobrar, como se observa en la figura 5.27.

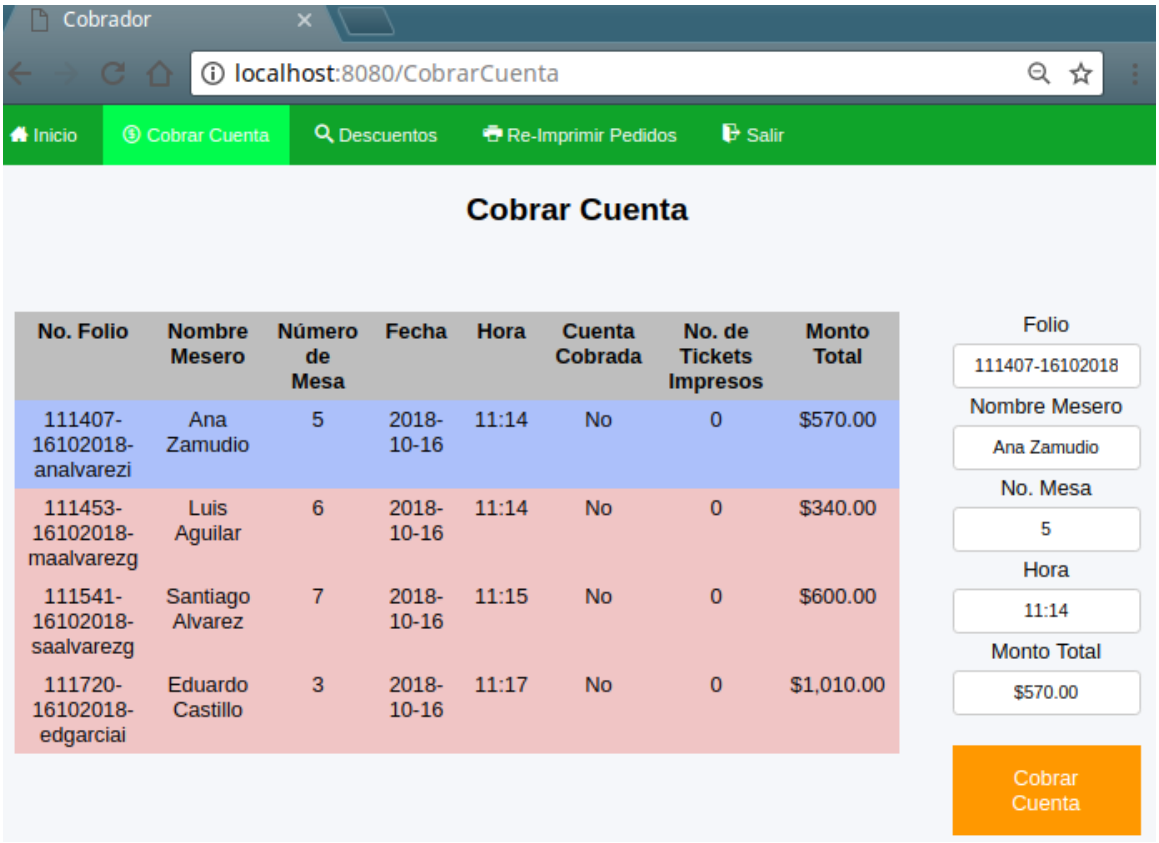


Figura 5.27 Cobrar cuenta cajero

Al cobrar la cuenta, esta automáticamente se bloquea para que el mesero ya no pueda continuar agregando más productos. Para validar este punto se accede a la cuenta del mesero y se puede observar que el botón de *Agregar Productos*, queda bloqueado, como se ilustra en la figura 5.28

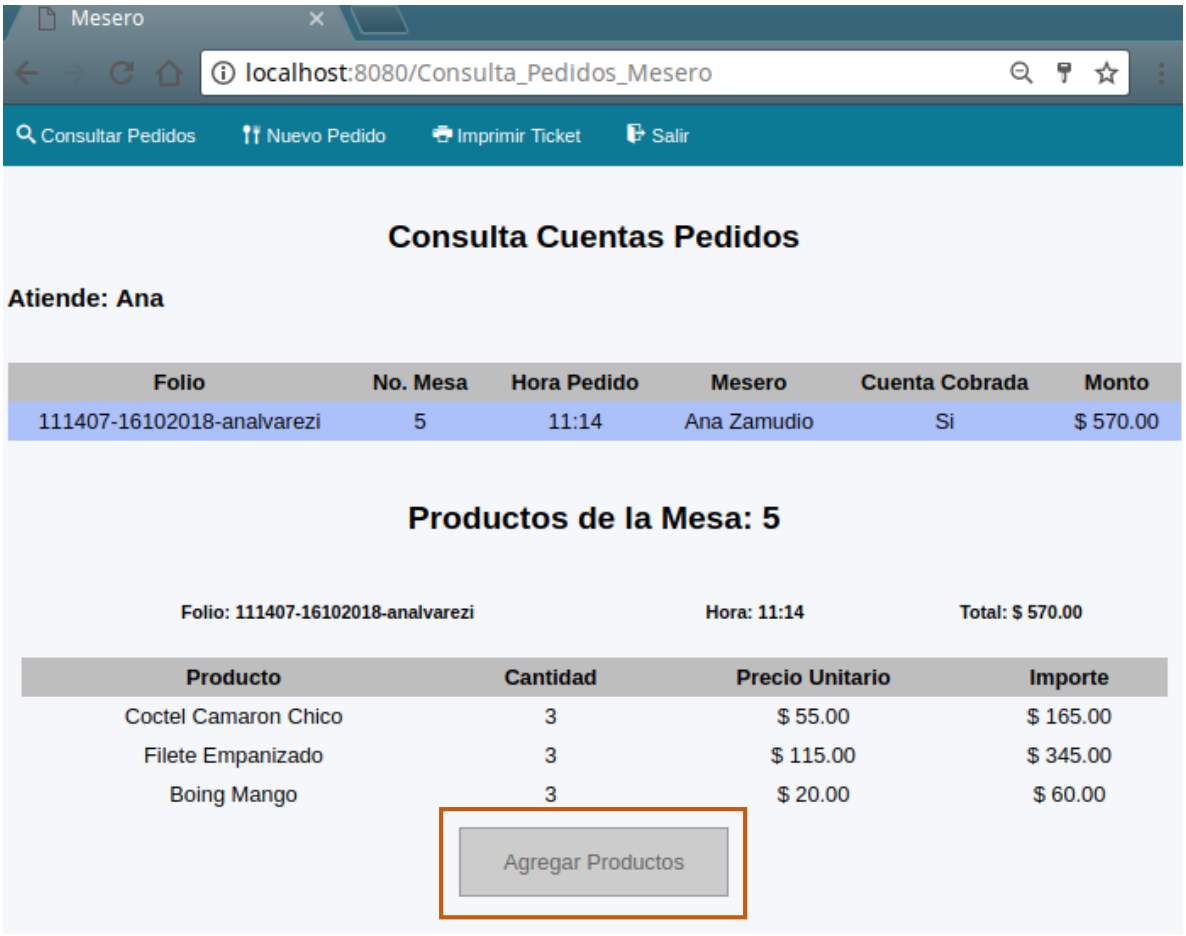


Figura 5.28 Cuenta bloqueada después de cobrar

- Aplicar descuento a una cuenta

Para realizar un descuento se accede a la sección de *Descuentos* y se elige la cuenta a la que se le va aplicar un descuento, como se observa en la figura 5.29

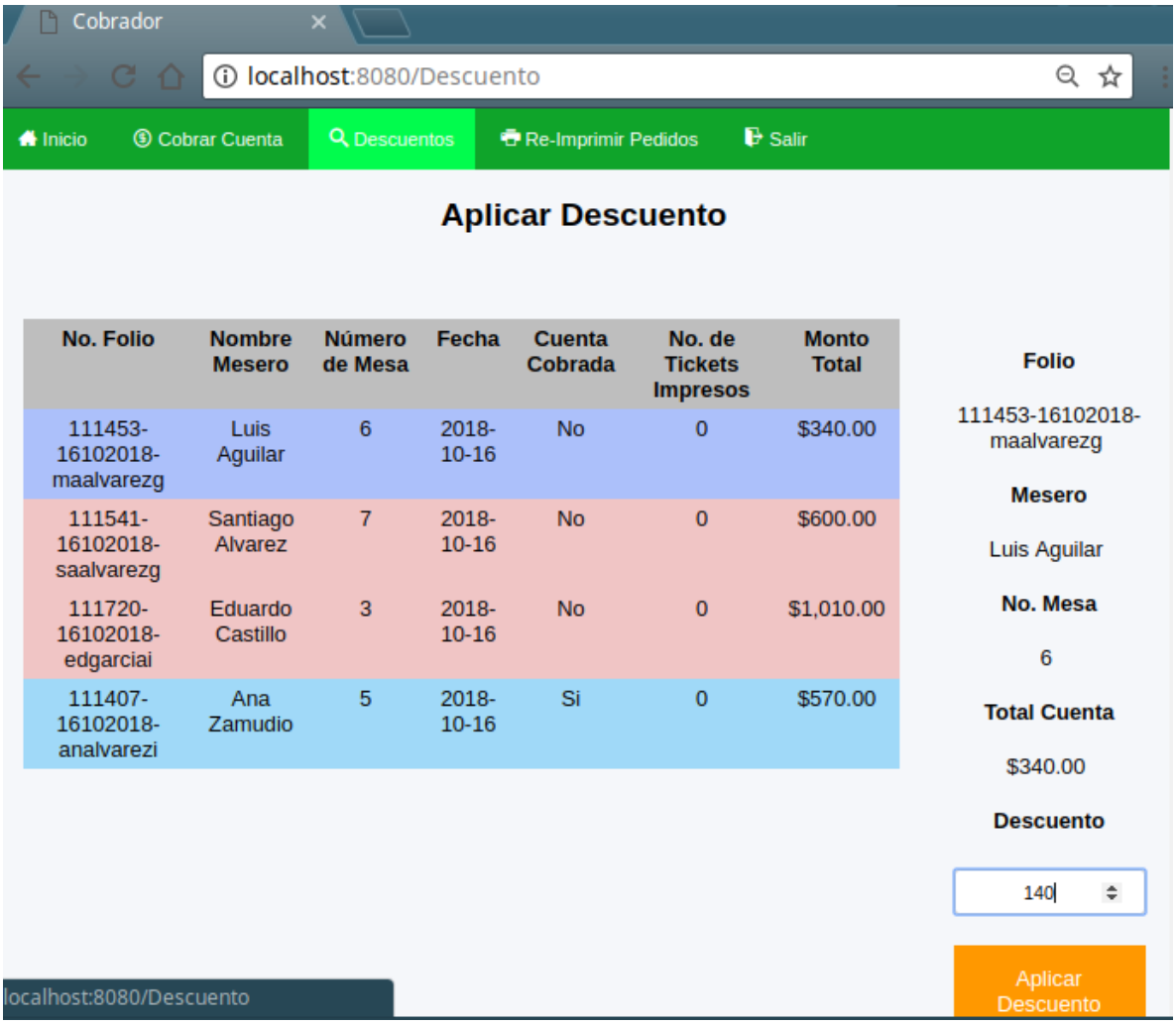


Figura 5.29 Aplicar descuento

Al aplicar el descuento este se verá reflejado inmediatamente, como se aprecia en la figura 5.30.

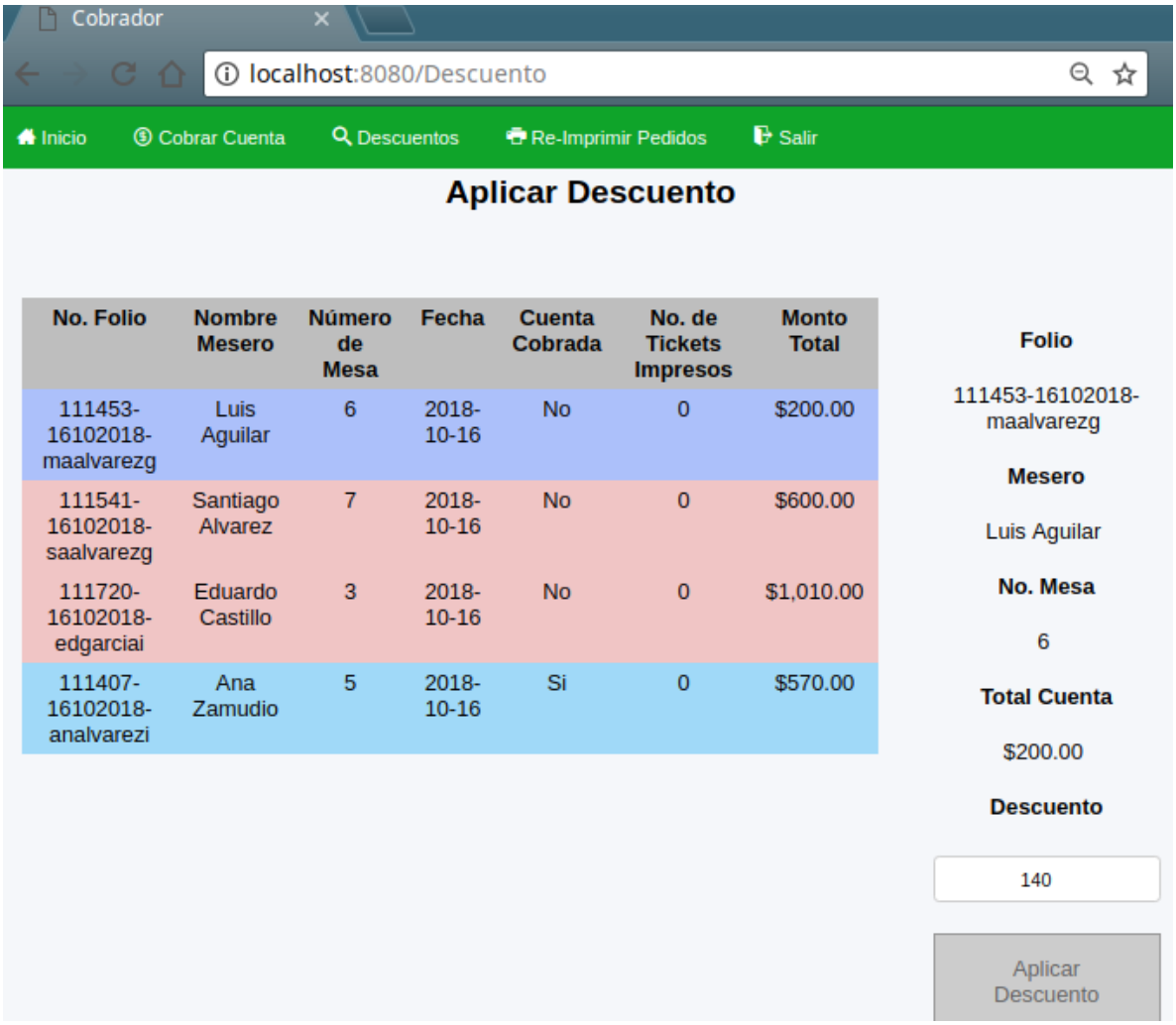


Figura 5.30 Descuento reflejado en la cuenta

Al imprimir el ticket del total de la cuenta también se verá reflejado el descuento, ilustrado en la figura 5.31

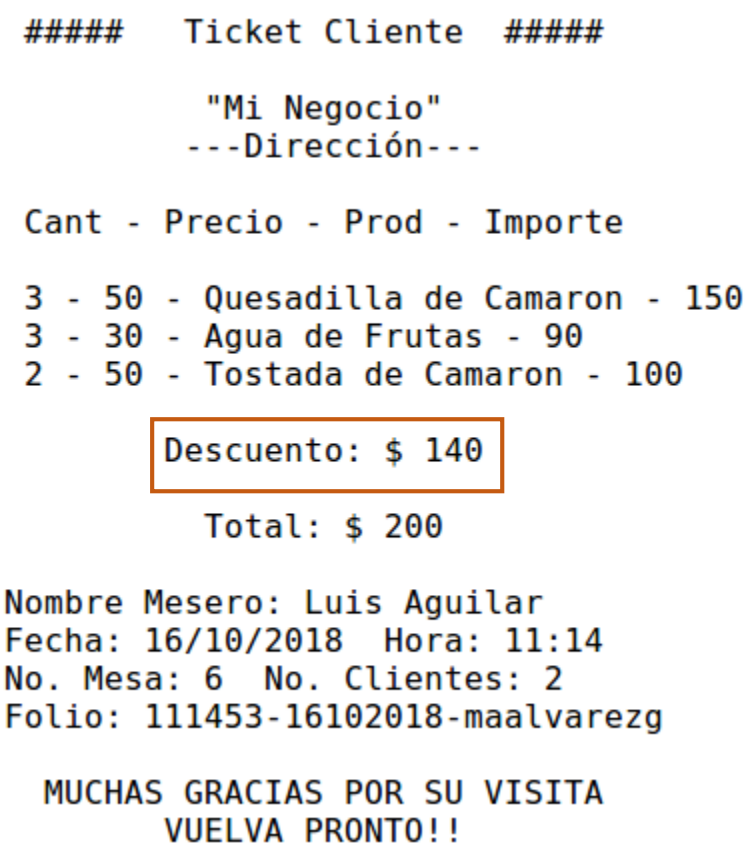


Figura 5.31 Ticket con descuento

- Re-imprimir un pedido del mesero

Para reimprimir los pedidos se accede a la sección del perfil del cajero, donde se tiene que elegir la fecha y el tipo de barra para elegir el ticket que se requiere reimprimir el pedido, como se observa en la figura 5.32

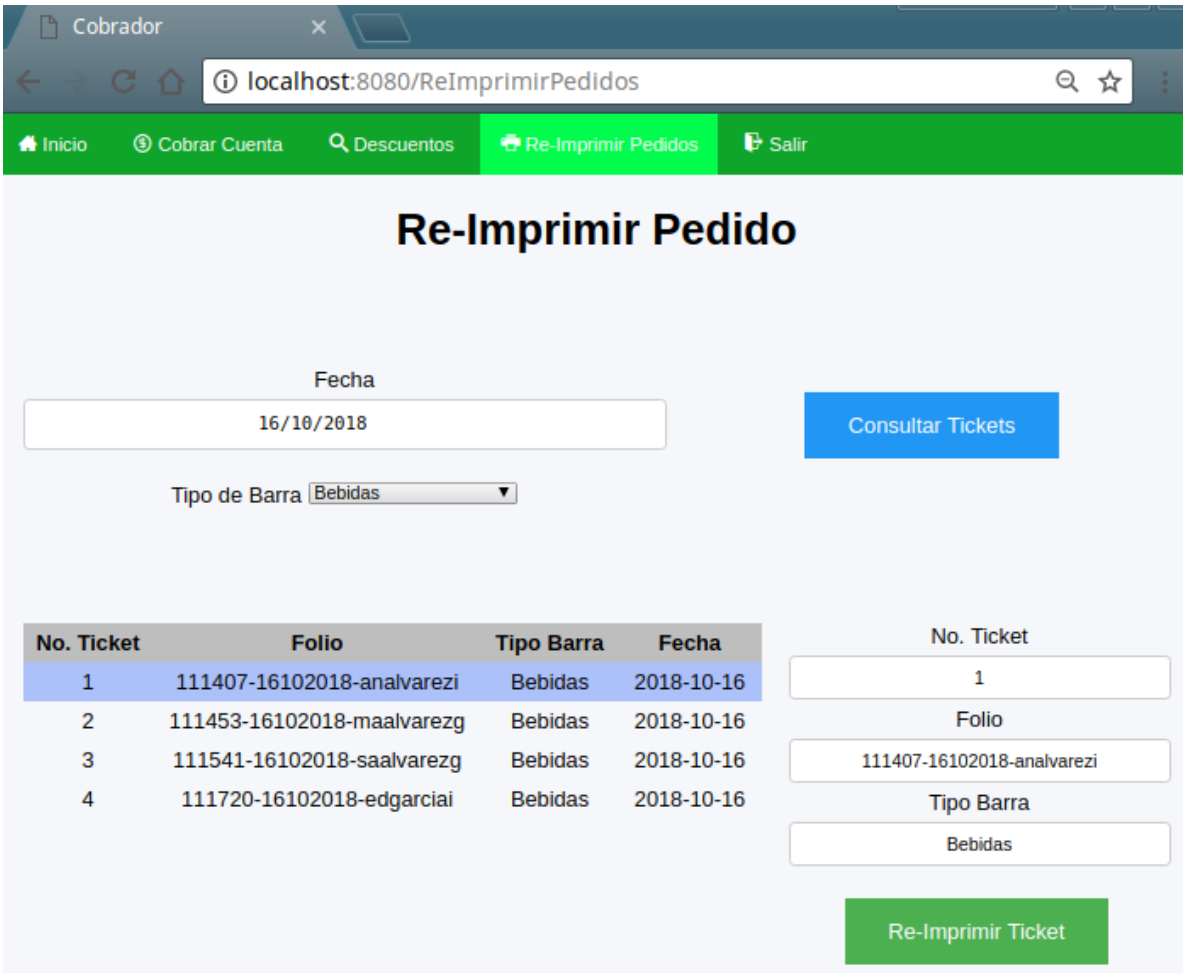


Figura 5.32 Re-imprimir pedido

Se tiene que oprimir el botón de *Re-Imprimir Ticket* para realizar la impresión del ticket, como se ilustra en la figura 5.33.

```
### BEBIDAS - No.Ticket - 1 ###

 Cantidad - Productos - Observaciones

 3 - Boing Mango -  N/A

Nombre Mesero: Ana Zamudio
Folio: 111407-16102018-analvarezi
Hora Pedido: 11_14_42
No. Mesa: 5
```

Figura 5.33 Ticket de re-impresión

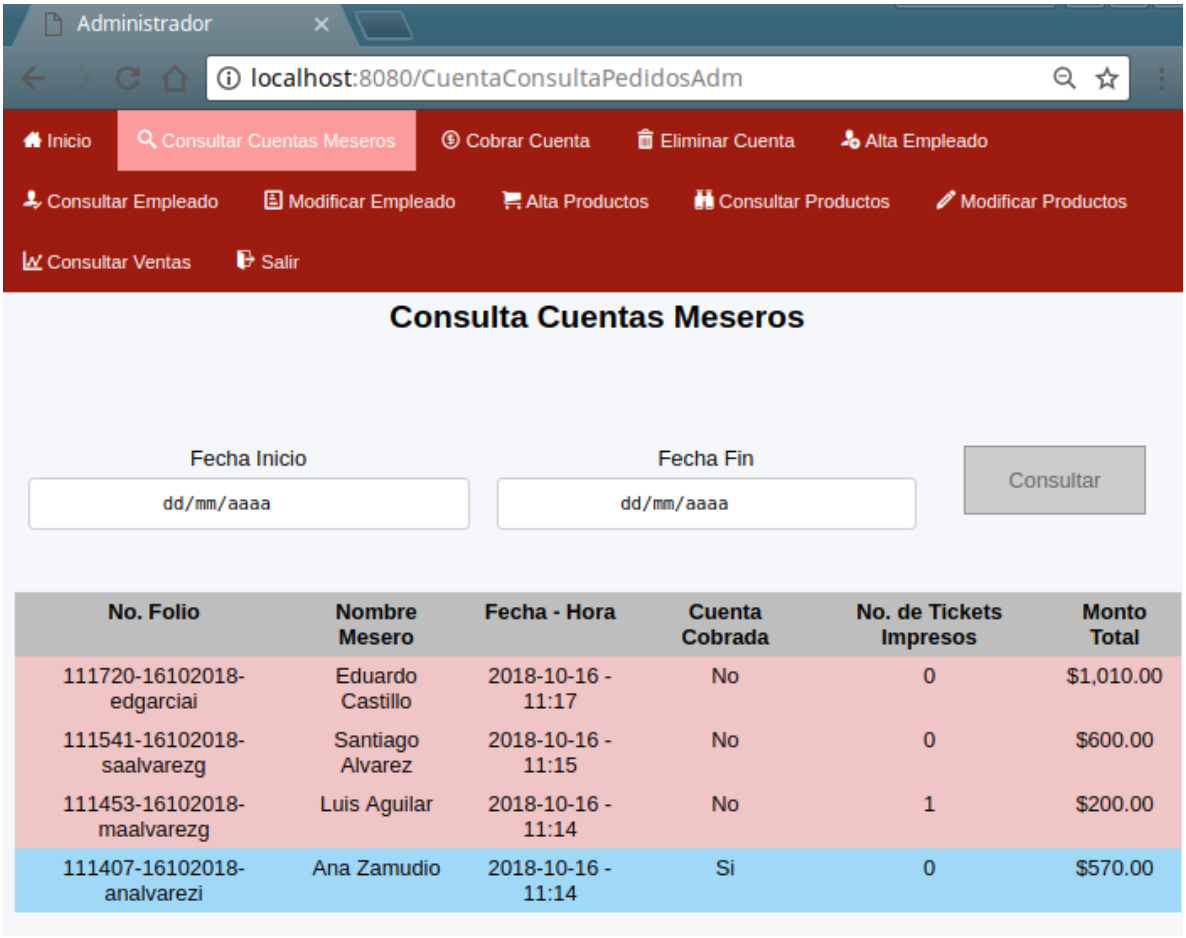
5.2.4 Pruebas perfil administrador

En el perfil del administrador se probaran los siguientes puntos:

- Consultar cuentas

Para consultar las cuentas se tiene que acceder al perfil del administrador en la sección de *Consultar Cuentas Meseros*, por default se muestran las cuentas del día actual, pero también se puede elegir dentro de un rango de fechas.

Las cuentas están ordenadas, en la parte superior se encuentran las cuentas que continúan sin cobrar y en la parte inferior las que ya están cobradas y cerradas, como se observa en la figura 5.34



No. Folio	Nombre Mesero	Fecha - Hora	Cuenta Cobrada	No. de Tickets Impresos	Monto Total
111720-16102018-edgarciai	Eduardo Castillo	2018-10-16 - 11:17	No	0	$1,010.00
111541-16102018-saalvarezg	Santiago Alvarez	2018-10-16 - 11:15	No	0	$600.00
111453-16102018-maalvarezg	Luis Aguilar	2018-10-16 - 11:14	No	1	$200.00
111407-16102018-analvarezi	Ana Zamudio	2018-10-16 - 11:14	Si	0	$570.00

Figura 5.34 Consulta cuentas meseros

- Alta, baja y modificación de un usuario

Para dar de alta un nuevo usuario en el sistema se accede dentro del menú a *Alta Empleado* y se ingresan los datos que se solicitan, como se observa en la figura 5.35

Figura 5.35 Alta usuario

Para validar que el usuario se haya registrado en el sistema se accede al menú en la sección de *Consultar Empleado*, como se observa en la imagen 5.36

Figura 5.36 Consulta empleados

Si se requiere de modificar los datos de contacto de un usuario, cambiar la contraseña o eliminar por completo al usuario del sistema, se accede a la sección de *Modificar Empleado,* como se aprecia en la figura 5.37.

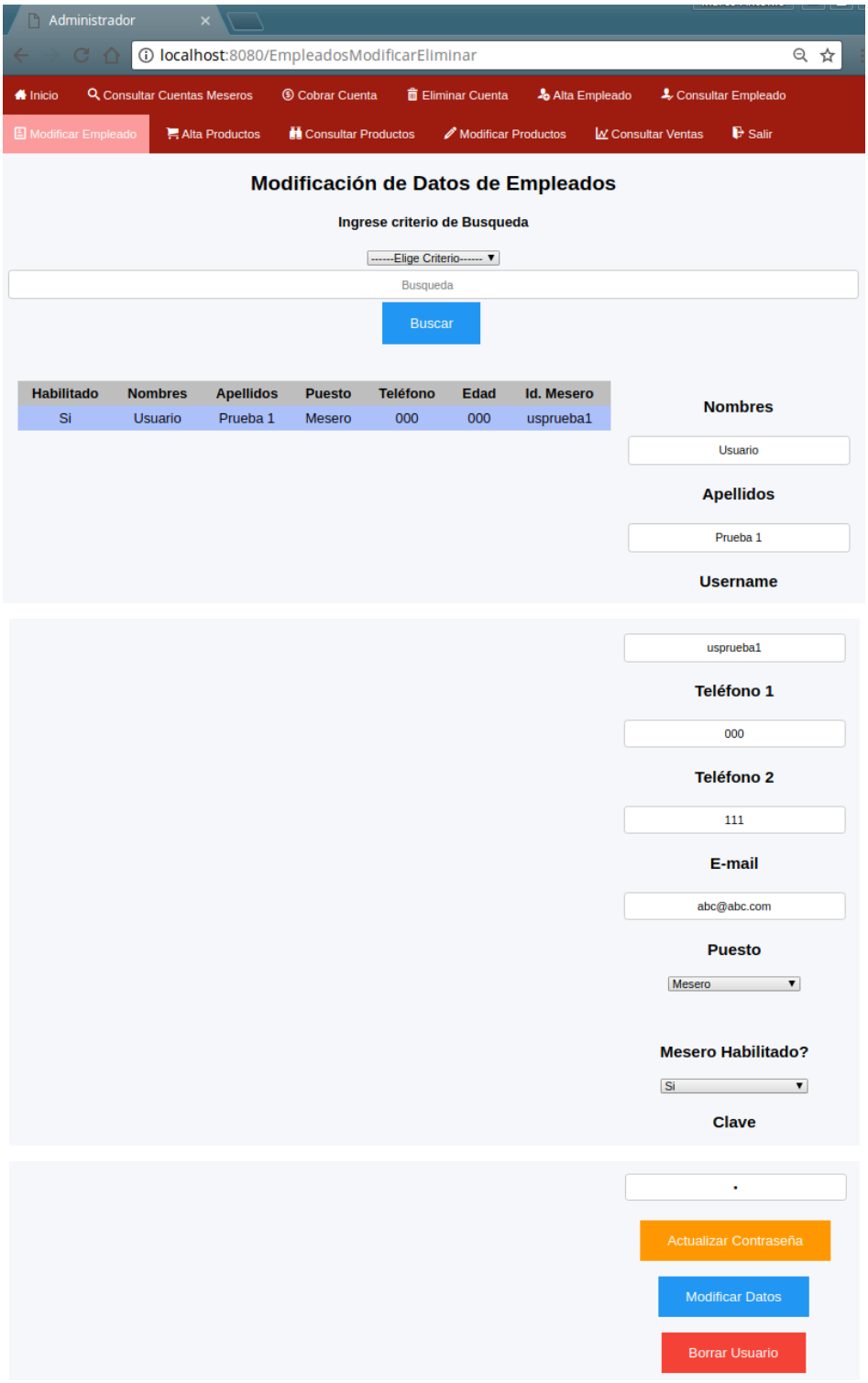


Figura 5.37 Actualización o borrado de usuarios

- Alta, baja y modificación de productos

Para dar de alta un producto nuevo se accede a la sección de *Alta Productos* y se ingresan los datos que solicita el sistema, como se parecía en la figura 5.38

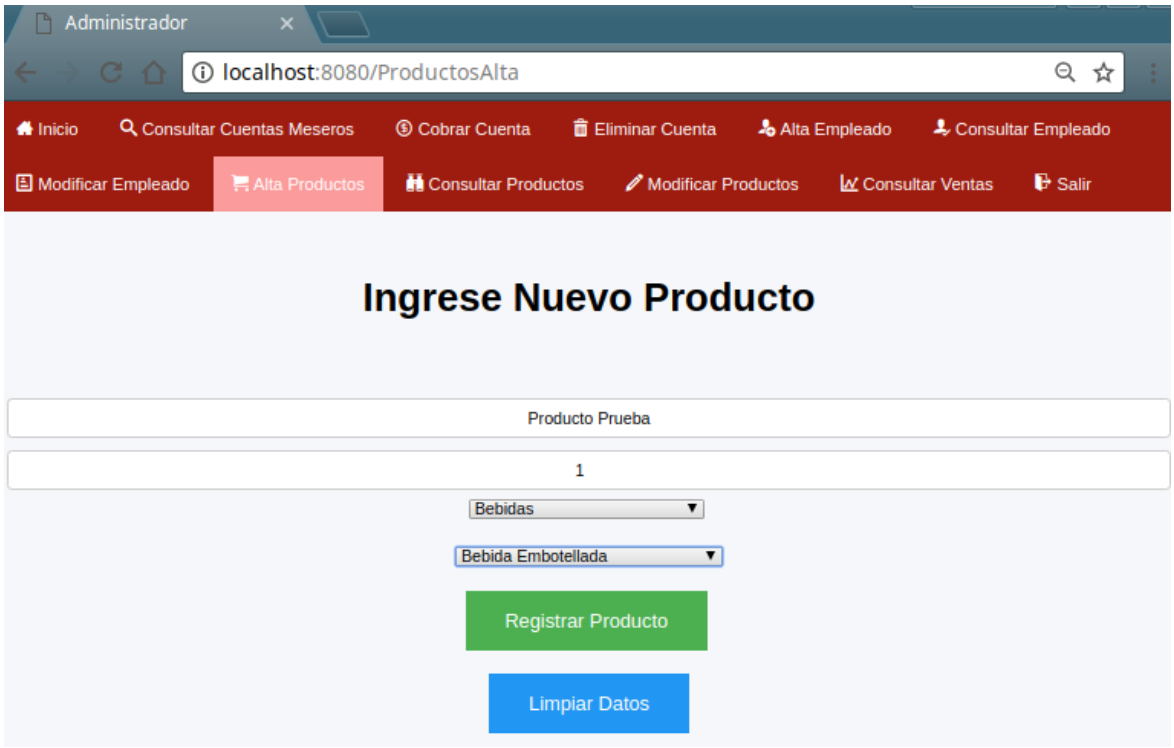


Figura 5.38 Alta productos

Para validar que efectivamente este dado de alta el producto se ingresa a la sección de *Consultar Productos*, como se aprecia en la imagen 5.39

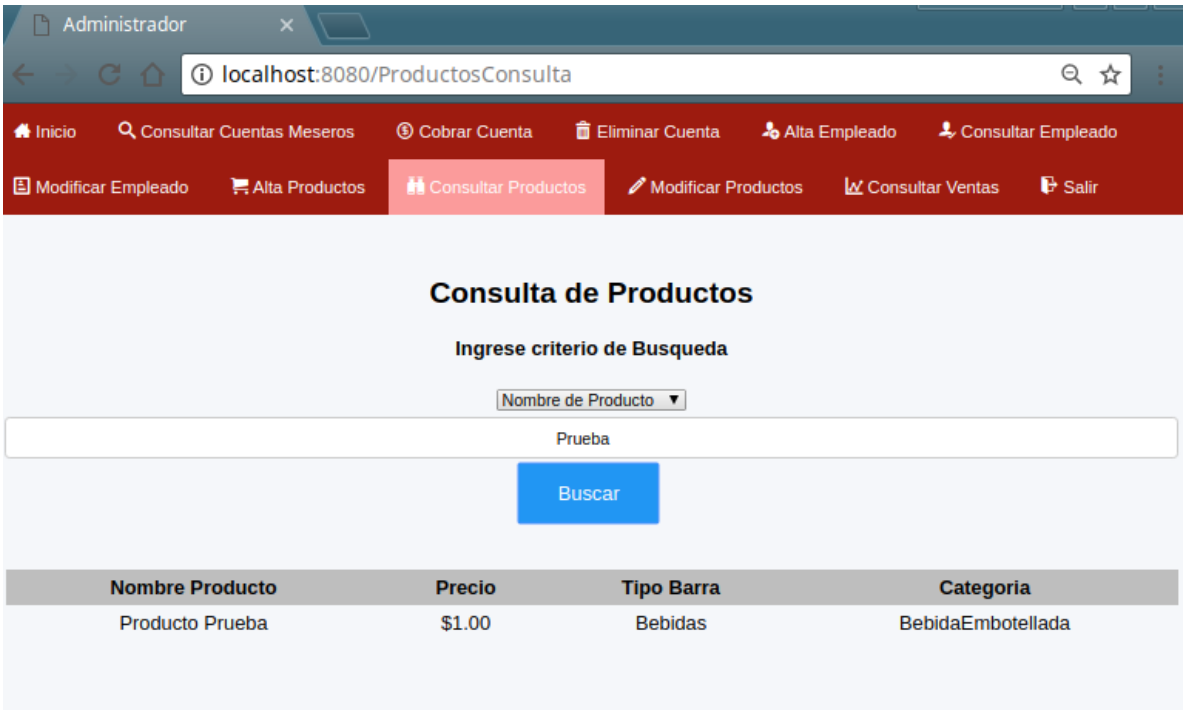


Figura 5.39 Consulta de productos

En caso de requerir modificar o eliminar por completo el producto se accede a la sección de Modificar Productos, donde se puede elegir entre modificar los datos del producto o eliminar por completo el producto, como se observa en la figura 5.40

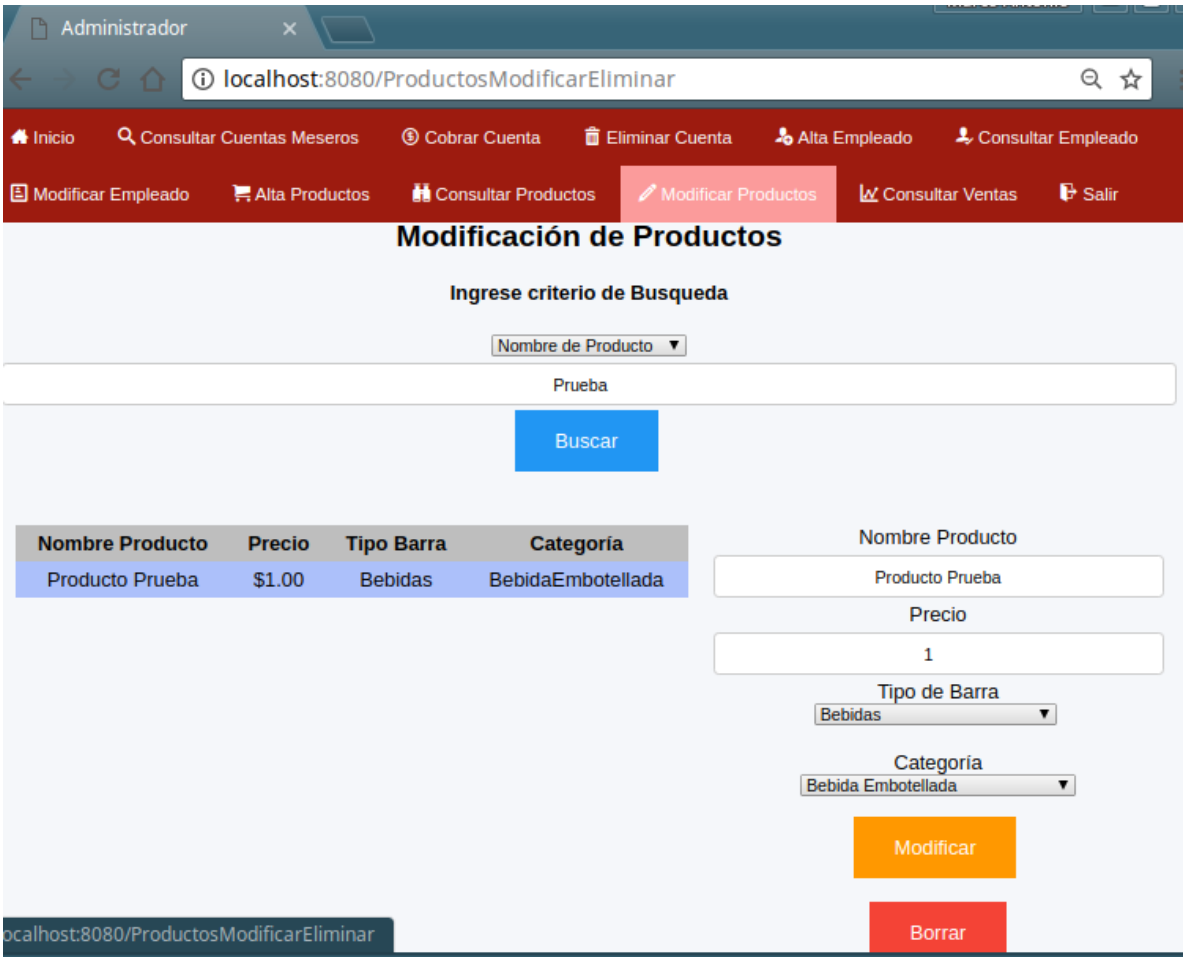


Figura 5.40 Actualización o borrado de productos.

- Reportes

Para generar el reporte de las ventas, se tiene que acceder a la sección de *Consultar Ventas* y elegir una fecha para visualizar el reporte, como se observa en la figura 5.41

Consulta Ventas por Día

Fecha Reporte

16/10/2018

Generar Reporte

Ventas Totales Del Día

Cuentas Cobradas	Descuentos	Total
Si	$ 140.00	$ 2,380.00

Ventas Por Mesero

ID_Mesero	Nombre Mesero	Cuenta Cobrada	Descuento por Mesero	Monto Total
edgarciai	Eduardo Castillo	Si	$ 0.00	$ 1,010.00
saalvarezg	Santiago Alvarez	Si	$ 0.00	$ 600.00
analvarezi	Ana Zamudio	Si	$ 0.00	$ 570.00
maalvarezg	Luis Aguilar	Si	$ 140.00	$ 200.00

Cantidad Cobrada por Cajero

ID_Cajero	Nombre Cajero	Monto Total
marcog	Marco Antonio	$ 1,810.00
edalvarezi	Eduardo	$ 570.00

Descuentos Aplicados

ID Usuario Aplico Descuento	Nombre Aplico Descuento	Total Descuentos
edalvarezi	Eduardo	$ 140.00

Productos Más Vendidos

Nombre Producto	Precio	Cantidad	Total de la Venta
Boing Mango	$ 20.00	6	$ 120.00
Filete Empanizado	$ 115.00	6	$ 690.00
Tostada de Camaron	$ 50.00	5	$ 250.00
Coctel Camaron Chico	$ 55.00	3	$ 165.00
Quesadilla de Camaron	$ 50.00	3	$ 150.00
Boing Guayaba	$ 20.00	3	$ 60.00
Coca Cola	$ 20.00	3	$ 60.00
Coctel Camaron Grande	$ 125.00	3	$ 375.00
Agua de Frutas	$ 30.00	3	$ 90.00
Enchiladas con Arrachera	$ 130.00	2	$ 260.00

Figura 5.41 Reportes por día

5.3 Análisis de resultados

En esta sección se detallan los resultados que se obtuvieron por la recolección de datos, así como el análisis del funcionamiento y rendimiento de la aplicación.

El periodo de prueba fue de 6 meses que comprende del mes de marzo del 2018 a agosto del 2018, con lo que se obtuvieron los siguientes datos:

- Cantidad de clientes atendidos durante el periodo: 4,334.
- Promedio de clientes atendidos por día: 24.
- Cantidad de platillos y bebidas despachados durante el periodo: 12,091
- Los 5 productos más vendidos:
    1. Coca cola – 842
    2. Tarro con chamoy – 532
    3. Cerveza victoria mega – 428
    4. Coctel mediano de camarón – 400
    5. Cerveza corona mega – 358
- Cantidad total de pedidos por hora, figura 5.42:

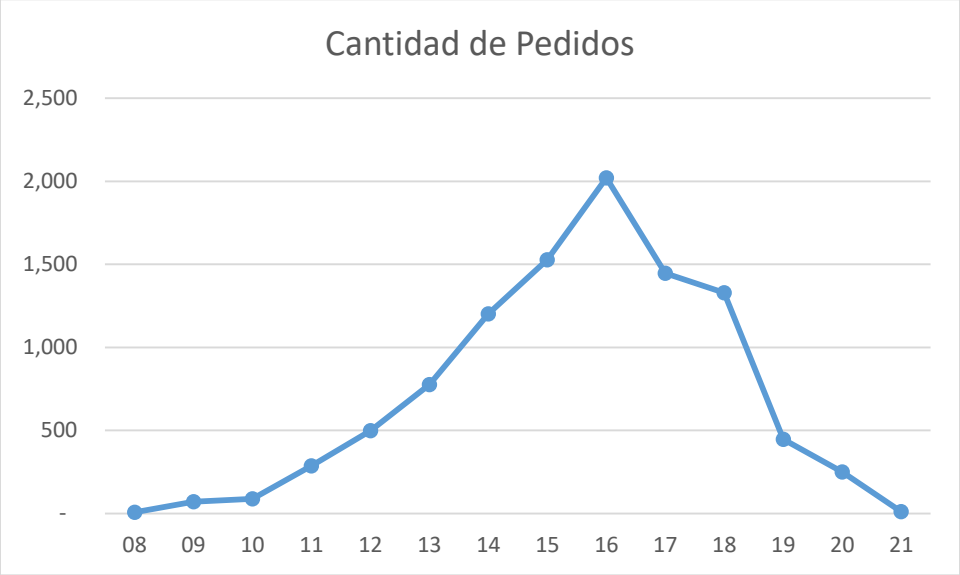


Figura 5.42 Cantidad de pedidos por hora

Antes de implementar el sistema era muy complicado saber cuál era la cantidad exacta de las ventas, ya que no se tenía personal dedicado para elaborar reportes sobres las ventas ni que llevara el control de los pedidos que realizaban los meseros.

El análisis de la venta del día se realizaba a mano y días después, lo que ocasionaba que no se detectaran oportunamente errores en la suma de las cuentas.

Esta situación se complicaba a un más cuando el flujo de clientes aumentaba en días festivos, únicamente se manejaban cifras aproximadas.

Al implementar el sistema se mejoró el control y la administración de los pedidos, ya se podía saber en tiempo real las ventas y las cuentas que están pendientes por cobrar, lo que conlleva a una mejor administración y aumento en las ganancias.

Capítulo 6
Mejora continua del proyecto

6. Mejora continua del proyecto

6.1 Áreas de oportunidad en el proyecto

Durante el desarrollo de este proyecto de tesis, sólo se cubrieron aspectos básicos del funcionamiento del sistema para cumplir con los propósitos planteados. Sin embargo aún se pueden incluir mejoras al sistema para optimizar su funcionamiento, entre las cuales se pueden destacar:

- **Desarrollo de aplicación nativa para los principales sistemas operativos en dispositivos móviles.**

Entre las principales mejoras que se pueden implementar en el sistema, es la creación de una aplicación móvil nativa para los principales sistemas como iOS y Android, su función principal sería, mejorar la interfaz gráfica del usuario, para aprovechar el espacio de la pantalla y agregar funcionalidades de notificación para los meseros.

Otra ventaja de la aplicación móvil, es que no se tiene que modificar la aplicación del lado del servidor, porque desde la aplicación móvil se lanzarían solicitudes HTTP en lugar de utilizar el explorador web, el servidor de aplicaciones recibe las solicitudes y las procesa. Para responder las solicitudes, el servidor envía la respuesta a la aplicación móvil y dependiendo de la respuesta, es como la aplicación móvil va a mostrar los datos en pantalla.

El intercambio de información entre el servidor de aplicaciones y la aplicación móvil es por medio de archivos en formato JSON, por lo que no se requiere de una aplicación intermediaria para hacer el intercambio de información.

- **Mejorar la implementación del módulo de inventarios y recetarios**

Otro de los aspectos que se tiene que mejorar es el control de los inventarios, la idea es implementar un módulo de alertas. Con base a los pedidos que los meseros generan, los productos de los pedidos se descuentan automáticamente del registro del almacén, ya sea que se descuente por unidad o a granel, en cuanto se rebase el límite del umbral definido, el sistema tiene que lanzar mensajes de alerta a los administradores para informar sobre la situación de desabasto en el almacén y así poder llamar oportunamente al proveedor para surtir más mercancía.

Para la sección de recetarios, es el módulo donde se almacena la información de los ingredientes y la cantidad que se debe utilizar para preparar cada platillo del menú, esta sección es importante para los cocineros ya que es una guía de cómo preparar los platillos y bebidas.

- **Mostrar pedidos en pantalla**

En lugar de imprimir los tickets de los pedidos, éstos se van mostrando en la pantalla del dispositivo ya sea teléfono celular o tablet, para que los cocineros preparen los pedidos conforme se están recibiendo, si es que algún cocinero tiene dudas de cómo elaborar el platillo, puede dar click en el pedido para abrir la sección de ingredientes y así el cocinero pueda visualizar como elaborar el platillo.

Cuando el platillo ya está listo, el cocinero marca el platillo como elaborado y pasa a elaborar el siguiente platillo.

- **Análisis y minería de datos**

Con la información recopilada, se puede realizar un análisis exhaustivo sobre la información, como por ejemplo, saber en qué días y que hora hay más actividad, que platillo consumen más los clientes y aplicar algoritmos predictivos y analíticos.

- **Funcionamiento remoto por medio de internet**

El sistema tiene como finalidad trabajar sobre una red local, sobre todo porque es más rápido y seguro, sin embargo también se puede implementar el sistema para que funcione de forma remota, es decir, desde un servidor central en internet, los clientes se pueden conectar para trabajar con el sistema de punto de venta.

Basta con que el cliente acceda a su cuenta y el sistema tendrá la capacidad de determinar a qué grupo de trabajo pertenece la cuenta, de esta forma se ofrecerían servicios en la nube, aunque la desventaja seria la velocidad y la disponibilidad, también esto implica volver al sistema más robusto y escalable.

Capítulo 7
Conclusiones

7. Conclusiones

Entre las principales ventajas de utilizar el framework Node.js, es la facilidad para desarrollar aplicaciones web escalables y el poco tiempo que se tiene que invertir para desarrollar una aplicación compleja, los factores que hacen esto posible es el lenguaje de programación JavaScript, que se utiliza tanto del lado de servidor, como del lado del cliente.

Sus características asíncronas hacen que la aplicación sea eficiente y liviana, ya que no requiere que se genere una gran cantidad de hilos de ejecución para procesar las solicitudes entrantes, éstas se delegan al sistema operativo o la base de datos. Las tareas se ejecutan en segundo plano, sin que se bloquee la aplicación. Cuando la tarea termina de ejecutarse se devuelve la respuesta y dependiendo si terminó en error o de forma correcta, posteriormente se muestra el resultado al usuario.

El sistema procesa las solicitudes realizadas en tiempo real, varios meseros pueden utilizar la aplicación de manera simultánea, sin que ésta se bloquee, además no se consumen demasiados recursos, como memoria y procesador, por lo que el servidor funciona muy bien en equipos de bajos recursos.

Node.js trabaja junto con la base de datos MongoDB, al utilizar el formato de archivos JSON, facilita el intercambio de datos entre la base de datos y el framework AngularJS que es la parte grafica que se encarga de mostrar los datos a los usuarios.

Otra de las razones de utilizar la base de datos MongoDB, es la escalabilidad del esquema de datos, sobre todo pensando a futuro, si el cliente requiere que se realicen modificaciones a la aplicación, o si se desea implementar el sistema en otro negocio que requiera otras reglas y esquema de datos, el sistema tiene la flexibilidad de adaptarse a las nuevas necesidades solicitadas, debido a su tipo de esquema dinámico.

Cabe mencionar que la integración que existe entre MongoDB, Node.js y Angular.js es por medio de archivos JSON, ocupar el mismo formato de archivos, facilita el almacenamiento, traslado y despliegue de datos.

El propósito principal de desarrollar el sistema de punto de venta para restaurantes, fue el de ayudar a las micro, pequeñas y medianas empresas que tienen dificultades en su administración.

Capítulo 7. Conclusiones

Al ser empresas pequeñas cuentan con poco capital monetario para invertir en sistemas robustos y complejos, además en su mayoría, los dueños poseen pocos o nulos conocimientos técnicos sobre el manejo de computadoras, el objetivo del sistema es dar soluciona a esta problemática implementando un sistema simple y eficiente en memoria que pueda funcionar sobre cualquier equipo de bajos recursos.

Después de implementar el sistema, en el negocio se observaron mejoras rápidamente, como lo son, mejoras en los pedidos, en la administración de los cobros, en la administración del personal, en el seguimiento de las cuentas para saber, cuántas mesas han sido atendidas por el mesero y cuántas mesas le faltaban por cobrar, pero sobre todo, se vio una mejora considerable en el aumento de las ganancias y se bajó el número de las perdidas monetarias.

Con el aumento en las ganancias, el negocio puede continuar invirtiendo y así mantenerse en constante crecimiento y no ser uno más en las estadísticas de los negocios que no dura más de 5 años por problemas administrativos y financieros.

Capítulo 8
Bibliografía y mesografía

8. Bibliografía y mesografía

- Alex R. Young, Marc Harter, Node.js in Practice-Meaning, 2015
- Dustin Metzgar, .NET Core in Action, Manning, 2016
- Adam Fowler, NoSQL For Dummies, John Wiley & Sons, Inc, 2015
- Gaurav Vaish, Getting Started with NoSQL, Packt Publishing, 2013
- Pramod J. Sadalage, Martin Fowler, NoSQL A Brief Guide to the Emerging Worl of Polyglot Persistence, Addison-Wesley, 2013
- Dan Sullivan, NoSQL for Mere Mortals, Addison-Wesley, 2015
- Kyle Banker, Peter Bakkum, Shaun Verch, Douglas Garrett, Tim Hawkins, MongoDB in Action, Second Edition, Manning, 2016
- Aleksa Vukotic, Nicki Watt, Tareq Abedrabbo, Dominic Fox, Jonas Partner, Neo4j in Action, Manning, 2015
- https://www.forbes.com.mx/pymes-mexicanas-un-panorama-para-2018/
- https://www.publimetro.com.mx/mx/noticias/2017/04/17/65-empresas-mexicanas-muere-cinco-anos.html
- https://nodejs.org/en/about/
- https://projectreactor.io/docs/core/release/reference/#intro-reactive
- http://www.baeldung.com/java-completablefuture
- http://www.uwanttolearn.com/android/pull-vs-push-imperative-vs-reactive-reactive-programming-android-rxjava2-hell-part2/
- https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html
- https://www.reactivemanifesto.org/
- https://profile.es/blog/que-es-la-programacion-reactiva-una-introduccion/
- https://docs.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/
- https://www.ibm.com/support/knowledgecenter/es/SSZJPZ_11.5.0/com.ibm.swg.im.iis.ds.basic.ddo/topics/r_dsbasic_Transaction_Properties.html
- http://guide.couchdb.org/editions/1/es/consistency.html
- https://blogs.oracle.com/spain/qu-es-una-base-de-datos-nosql
- https://es.slideshare.net/mercyo/bd-no-sql-conceptos-basicos
- https://aws.amazon.com/es/nosql/
- https://programtalk.com/java/java-9-new-features/

- https://blogs.oracle.com/spain/qu-es-una-base-de-datos-nosql
- https://docs.mongodb.com/manual/core/map-reduce/
- https://neo4j.com/docs/developer-manual/current/
- https://neo4j.com/developer/guide-data-modeling/
- https://neo4j.com/developer/graph-algorithms/
- https://blog.makeitreal.camp/lenguajes-compilados-e-interpretados/
- https://medium.com/laboratoria-developers/introducci%C3%B3n-a-la-programaci%C3%B3n-funcional-en-javascript-parte-1-e0b1d0b2142e