---


TP Trabajo práctico

**Materia:** Seminario De Práctica De Informática

**Año:** 2024

**Fecha de Entrega:** 30/06/2024

**AUTOR:** **Giuliano Danilo Kuhn**

**LEGAJO N°**: VINF011416

**DNI:** 44.369.822

---

# TITULO

**Desarrollo de un sistema de gestión y servicio moderno de atención al cliente, procesando datos para minimizar el tiempo de espera de los comensales.**

# INTRODUCCION

Comedor “Lo de María”, es un restaurante donde se ofrece Item sofisticada y fina elaborada a diario en su propia cocina.

Lo de María entiende que el tiempo de sus clientes es muy valioso, donde la exquisitez y la atención al detalle son principios fundamentales, por eso su prioridad es la excelencia, mejorando los tiempos de espera de los clientes en el restaurante y la calidad de sus platos.

Con el proyecto se pretende reducir los tiempos de espera, como así también evitar posibles errores, buscando hacer que la experiencia en el restaurante sea más fácil y agradable para nuestros clientes.

---

# JUSTIFICACION

El restaurante “Lo de María” cuenta con un gran número de clientes exigentes, pero su falta de organización es su limitante, aumentando el tiempo de espera.

El tiempo de espera se debe a que los mozos se confunden con las mesas que le corresponden a cada uno, discutiendo entre ellos cual le corresponde a cada uno, así también elevando el tiempo de espera para tomar una orden o para cobrar, como así también gente que se acumula para saber si hay disponibilidad de mesas en el lugar.

La implementación de un sistema de gestión y servicio en el restaurante requerirá mejoras tecnológicas para organizar y agilizar los procesos internos, como así también facilitar la experiencia del cliente y evitar errores.

Como el sector de la gastronomía es muy competitivo, la implementación de este sistema de gestión y servicio proporcionaría una experiencia más satisfactoria, lo que contribuiría a una distinción dentro del negocio culinario, promoviendo la fidelización y crecimiento del restaurante.

# DEFINICIONES DEL PROYECTO

## Objetivo general del proyecto

Desarrollar un sistema de gestión y administración de pedidos y asignación de mesas, para así reducir el tiempo de espera de los clientes como así también evitar errores (que derivan en más tiempos de esperas para resolver esos problemas) maximizando el disfrute del cliente.

# Definiciones del sistema

## Objetivo general del sistema

Recolectar y analizar datos para optimizar el tiempo de los clientes en el restaurante y facilitando su estadía en el local.

---

# Elicitación:

Para este proyecto, se definió realizar encuestas, entrevistas, y observación, técnicas que permitieron recopilar la información esencial para luego redactar los requerimientos.

**Entrevistas:** se utilizaron para obtener datos y una comprensión inicial de las necesidades y expectativas de las partes interesadas (el que quiere y paga el desarrollo), para ello se elaboran entrevistas abiertas.

La entrevista va a tener como bloque temático saber cuál es el problema que está teniendo las partes interesadas, haciendo preguntas abiertas sobre las necesidades que tiene. De los problemas que indique el entrevistado, se indagará mas sobre las posibles causas de los problemas, siguiendo con preguntas abiertas para que el entrevistado, nos cuente lo máximo posible.

**Observaciones:** se decide ir un día sin un aviso de antemano al restaurante para observar un día corriente en el restaurante, y así poder identificar si las causas que nos indicó el entrevistado son las causas reales e identificar otros posibles problemas.

Esto se haría observando el funcionamiento de las partes involucradas en el problema que nos dijo el entrevistado.

**Encuestas:** se utilizaron para obtener y profundizar en la información obtenida para una comprensión más detalla de las necesidades de los distintos usuarios.

En particular se van a entrevistar a mozos, clientes y cajeros; con preguntas cerradas sobre el funcionamiento de la atención hacia los clientes del local. Y con una pregunta abierta de una sugerencia obligatoria para mejorar sobre el mismo tema.

---

# CONOCIMIENTO DEL NEGOCIO

El sistema que se va a desarrollar debe permitir la gestión y administración del servicio de mozos hacia los clientes, como también debe evitar posibles errores y realizar un procesamiento de datos para optimizar el tiempo de espera de los clientes en el negocio.

En este local, los clientes si hay mucha gente, le preguntan a uno de los mozos, si hay una mesa disponible, en caso contrario solo van y se sientan en una mesa que este libre, luego el mozo va a la mesa para dejar la carta del restaurante, luego de un tiempo vuelve a la mesa para tomar la orden, este mozo tiene que luego llevar la orden a la caja para que la caja lleve un recuento de lo que se va pidiendo en cada mesa, que luego pasa a la cocina para ser preparada.

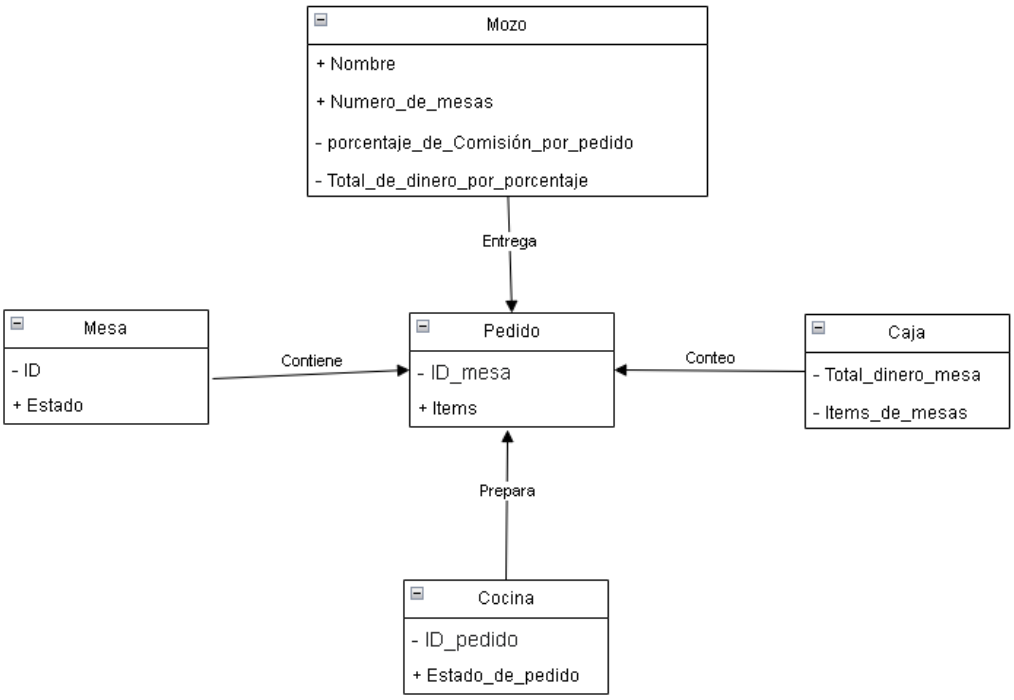
Como cualquier restaurante el mozo puede tener varias mesas él solo. Por mesa atendida el mozo gana un porcentaje de dinero extra.

Por su parte la caja lleva un recuento de lo que pidió cada mesa y los precios.

Cada mesa tiene un número que se le es entregado cuando se le trae la carta del menú.

---

# DIAGRAMA DE DOMINIO



---

# PROPUESTA DE SOLUCIÓN

Una vez que se ha realizado el diagnóstico, se presenta una propuesta de solución funcional, la propuesta técnica, la arquitectura planteada para el despliegue y la lista de requerimientos a incluir en el sistema.

## Propuesta funcional

Se propone desarrollar un sistema de gestión y administración, no solo para favorecer la eficiencia y rentabilidad del restaurante, sino que también contribuirá a mejorar la experiencia general de los clientes del personal, con un enfoque modular y adaptable que permite su personalización según las necesidades especificas del negocio, garantizando así su efectividad a largo plazo.

Para ello Cada mesa tendrá un centro de mesa, que dirá a la página que se debe conectar para que puedan hacer la orden, o un código QR hacia la misma página.

Además que también dirá el código de la mesa y las instrucciones para hacer la orden, una vez hecha la orden, esta pasa al sistema para ser leída lo más rápido posible por el cocinero, y utilizando un algoritmo para la asignación de la mesa a mozos equitativamente.

## Propuesta técnica

El desarrollo del sistema de gestión y monitoreo de centrales eólicas se va a realizar en Java, que cuenta con una gran variedad de bibliotecas y *frameworks* [marcos de trabajo] que permiten el escalamiento. La persistencia se realizará en MySQL, que es una base de datos relacional abierta, flexible y de alto rendimiento para el almacenamiento de datos en tiempo real.

Se va a emplear JDBC (Java Database Connectivity) para consultar y realizar la persistencia de los datos desde Java. Esta API (interfaz de programación de aplicaciones) estándar permite a las aplicaciones Java interactuar con bases de datos.

La tecnología de comunicación a utilizar será la siguiente:

1. La conexión de red dentro de cada central se realizará mediante un cableado Ethernet, que proporciona una infraestructura confiable y eficiente.

---

# REQUERIMIENTOS

| Requerimiento No Funcional | Descripción |
|---|---|
| **RNF001** | -El lenguaje de programación será java. |
| **RNF002** | - El sistema debe contar con una base de datos MySQL. |
| **RNF 003** | - El sistema debe utilizar XAMPP como entorno de desarrollo web de código abierto. |
| **RNF 004** | - Para la persistencia y consulta de datos en la BD se debe utilizar un patrón MVC (modelo-vista-controlador). |
| **RNF 005** | - Debe estar instalado en el propio restaurante |
| **RNF 006** | - El sistema debe ser escalable |
| **RNF 007** | -El sistema necesita 3 pantallas para su funcionamiento |
| **RNF 008** | -Cada mesa contara con un Identificador |
| **RNF 009** | -EL sistema debe contar con un cartel para el número de mesas disponibles |

---

| Requisitos Funcionales | Descripción |
|---|---|
| **RF001** | El sistema debe identificar cada mesa mediante el ingreso de un código (ID) |
| **RF002** | El sistema debe mostrar la carta del lugar |
| **RF003** | Se debe poder seleccionar lo que se quiere consumir |
| **RF004** | El sistema debe poder calcular el precio total de la orden |
| **RF005** | El sistema debe generar una orden |
| **RF006** | El sistema debe mostrar orden y su estado en caja y en cocina |
| **RF007** | El sistema debe asignar las mesas equitativamente a cada mozo |
| **RF008** | El sistema debe asignar 2 estados (preparando y listo) a cada orden |
| **RF009** | El chef debe poder cambiar de estado a cada orden |
| **RF010** | El sistema debe mostrar a los mozos: las ordenes listas y la mesa a la que pertenecen y el mozo que le corresponde |
| **RF011** | El sistema debe permitir el ingreso mediante un usuario y contraseña |
| **RF012** | El sistema debe permitir al gerente agregar un nuevo mozo |
| **RF013** | Al agregar un nuevo mozo, este debe poder ingresar su usuario y contraseña |
| **RF014** | El sistema debe permitir al gerente quitar un mozo |
| **RF015** | El sistema debe permitir al gerente agregar una nueva mesa |
| **RF016** | Al agregar una nueva mesa, el sistema debe generar un nuevo ID para esa mesa |
| **RF017** | El sistema debe permitir al gerente quitar una mesa |
| **RF018** | El sistema debe permitir mostrar solo al gerente informes |
| **RF019** | El sistema debe registrar los pedidos |
| **RF020** | El sistema debe permitir al gerente agregar Ítems del menú |
| **RF021** | El sistema debe permitir al gerente quitar Ítems del menú |

---

# TRAZABILIDAD

| Requerimientos | Caso de uso | Actor principal | Paquete del análisis | Comentario |
|---|---|---|---|---|
| **RF01** | CU001 | Cliente | Cliente | Ingresar código de mesa |
| **RF02** | CU002 | Cliente | Cliente | Mostrar menú |
| **RF03** | CU002 | Cliente | Cliente | Seleccionar lo que se va a consumir |
| **RF04** | CU002 | Cliente | Cliente | Calcular total |
| **RF05** | CU003 | Cliente | Cliente | Genera la orden |
| **RF08** | CU003 | Cliente | Cliente | Asignar el estado “Preparando” |
| **RF19** | CU003 | Cliente | Cliente | Registrar el pedido |
| **RF07** | CU004 | Cliente | Cliente | El sistema Asigna mesa a un mozo |
| **RF11** | CU005 | Mozo | Mozo | Ingreso al trabajo mediante su usuario y contraseña |
| **RF13** | CU005 | Mozo | Mozo | Registrar usuario y contraseña para nuevo mozo |
| **RF10** | CU006 | Mozo | Mozo | Se muestra las mesas de cada mozo y sus ordenes con sus estados |
| **RF9** | CU007 | Chef | Chef | Cambia de estado las ordenes de pendiendo de si están listas o no |
| **RF6** | CU008 | Chef | Chef | Mostrar las ordenes y sus estados para poder cambiarlas de estado |
| **RF6** | CU008 | Caja | Caja | Mostrar las ordenes y sus estados para poder luego cobrar |
| **RF12** | CU009 | Encargado | Encargado | Agregar mozo |
| **RF14** | CU010 | Encargado | Encargado | Quitar mozo |
| **RF15** | CU011 | Encargado | Encargado | Agregar mesa |
| **RF16** | CU011 | Encargado | Encargado | Se genera nuevo ID de la mesa cuando se agrega una nueva |
| **RF17** | CU012 | Encargado | Encargado | Quitar mesa |
| **RF20** | CU013 | Encargado | Encargado | Agregar Ítem |
| **RF21** | CU014 | Encargado | Encargado | Quitar Ítem |
| **RF18** | CU015 | Encargado | Encargado | Generar informes |
| **RF10** | CU0016 | Mozo | Consulta | Consultar ordenes a la base de datos |
| **RF6** | CU0016 | Mozo | Consulta | Consultar ordenes a la base de datos |
| **RF10** | CU0016 | Chef | Consulta | Consultar ordenes a la base de datos |
| **RF6** | CU0016 | Chef | Consulta | Consultar ordenes a la base de datos |
| **RF10** | CU0016 | Caja | Consulta | Consultar ordenes a la base de datos |
| **RF6** | CU0016 | Caja | Consulta | Consultar ordenes a la base de datos |

---

| | |
|---|---|
| CU001 | Iniciar orden |
| CU002 | Mostrar menú |
| CU003 | Generar orden |
| CU004 | Asignar mesa a mozo |
| CU005 | Ingresar usuario y contraseña |
| CU006 | Mostrar mesas, ordenes y mozos |
| CU007 | Cambiar de estado |
| CU008 | Mostrar orden y sus estado |
| CU009 | Agregar mozo |
| CU010 | Quitar mozo |
| CU011 | Agregar mesa |
| CU012 | Quitar mesa |
| CU013 | Agregar Ítem |
| CU014 | Quitar Ítem |
| CU015 | Generar informes |
| CU016 | Consultar orden |
| CU017 | Orden a cartel |

---

# DIAGRAMA DE CASO DE USO

---

| | |
|---|---|
| **Caso de uso** | CU001 Iniciar orden |
| **Actores** | Cliente |
| **Referencias** | **RF01** |
| **Descripción** | El cliente introduce el código de la mesa en su interfaces |
| **Precondición** | |
| **Flujo principal** | 1- El cliente entra a la interfaz de cliente mediante entrando un link o escaneando un código qr que estará en la mesa<br>2- El sistema muestra un formulario vacio para agregar el código de la mesa<br>3- El cliente completa dicho formulario<br>4- El cliente oprime el botón ver menú<br>5- El sistema valida si dicho código es correcto (S5) |
| **Postcondición** | El sistema pasa al CU002 |
| **Flujo alternativo** | S5- El código es incorrecto: 1- muestra un mensaje de error |
| **Excepciones** | No contempla |

---

| | |
|---|---|
| **Caso de uso** | CU002 Mostrar menú |
| **Actores** | Cliente |
| **Referencias** | **RF02, RF03, RF04** |
| **Descripción** | Muestra toda la carta del restaurante al cliente |
| **Precondición** | Haber pasado por el CU001 |
| **Flujo principal** | 1- El sistema muestra todo el menú del local<br>2- El cliente selecciona lo que va a consumir (S2)<br>3- El sistema va calculando en tiempo real el total |
| **Postcondición** | No contempla |
| **Flujo alternativo** | |
| **Excepciones** | No contempla |

---

| | |
|---|---|
| **Caso de uso** | CU003 Generar orden |
| **Actores** | Cliente, Administrador de datos |
| **Referencias** | **RF05, RF08, RF019** |
| **Descripción** | El cliente confirma la orden y el sistema la registra |
| **Precondición** | Haber seleccionado al menos 1 ítem del menú |
| **Flujo principal** | 1- El cliente en el menú le da a la opción <<generar orden>> (S1)<br><br>2- El sistema crea un registro de la orden<br><br>3- El sistema muestra un mensaje de confirmación, indicando que la orden ya paso a la cocina |
| **Postcondición** | El sistema pasa al CU004 |
| **Flujo alternativo** | S1-El cliente decide irse:<br><br>1- No oprime el botón <<generar orden>> y en 5 minutos la mesa se muestra como libre |
| **Excepciones** | No contempla |

---

| | |
|---|---|
| **Caso de uso** | CU004 Asignar mesa a mozo |
| **Actores** | Cliente, Administrador de datos |
| **Referencias** | **RF07** |
| **Descripción** | Le asigna ordenes a mozos equitativamente |
| **Precondición** | Haber pasado el CU003 |
| **Flujo principal** | 1- El sistema asigna la mesa y la orden equitativamente a uno de todos los mozos disponibles, en base a distintos aspectos:<br>• Número de mesas<br>• Número de mozos<br>• Número de ordenes por mesa |
| **Postcondición** | Distribución de mesas equitativa entre los mozos |
| **Flujo alternativo** | No contempla |
| **Excepciones** | No contempla |

---

| | |
|---|---|
| **Caso de uso** | CU005 Ingresar usuario y contraseña |
| **Actores** | Mozo, Chef, Encargado, Caja, Administrador de datos |
| **Referencias** | **RF11, RF13** |
| **Descripción** | Los empleados y gerente, pueden registrarse y/o ingresar al trabajo con un usuario y contraseña |
| **Precondición** | **Registrarse:** ser llamado del Agregar mozo **Ingresar:** haberse registrado |
| **Flujo principal** | 1- El sistema muestra un formulario vacio para crear ese usuario nuevo<br><br>2- El personal a registrar completa el formulario proporcionando la siguiente información:<br><br>• Nombre de Usuario<br>• Contraseña<br><br>3- El personal a registrar confirma la entra de datos<br><br>4-El sistema valida que el usuario y contraseña sean validos<br><br>5- El sistema muestra un mensaje de confirmación, indicando que se inicio sesión correctamente |
| **Postcondición** | El Personal inicia sección/ se registra correctamente |
| **Flujo alternativo** | En caso de que a este caso de uso, lo llame el caso de uso Agregar mozo:<br><br>1- El sistema muestra un formulario vacio para crear ese usuario nuevo<br><br>2- El personal a registrar completa el formulario proporcionando la siguiente información:<br><br>• Nombre de Usuario<br>• Contraseña<br>• Repetir Contraseña<br><br>3- El personal a registrar confirma la entra de datos<br><br>4-El sistema valida que el usuario no esté ocupado<br><br>5- El sistema Agrega la persona en su rol correspondiente<br><br>6- El sistema muestra un mensaje de confirmación, indicando que el registro se logro con éxito |
| **Excepciones** | No contempla |

---

| | |
|---|---|
| **Caso de uso** | CU006 Mostrar mesas, ordenes y mozos |
| **Actores** | Mozo, Administrador de datos |
| **Referencias** | **RF10** |
| **Descripción** | Muestra las mesas, ordenes, estado de cada orden y mozos para la organización de los mozos |
| **Precondición** | |
| **Flujo principal** | 1-El sistema llama a CU016<br><br>2- El sistema muestra en una interfaz de mozos, las mesas con sus respectivas órdenes. Esto ordenado por mozo. |
| **Postcondición** | No contempla |
| **Flujo alternativo** | No contempla |
| **Excepciones** | No contempla |

---

| | |
|---|---|
| **Caso de uso** | CU007 Cambiar de estado |
| **Actores** | Chef, Administrador de datos |
| **Referencias** | **RF08, RF09** |
| **Descripción** | Permite al chef cambiar de estado las ordenes cuando estén listas o no |
| **Precondición** | Haber una orden activa |
| **Flujo principal** | 1- El chef oprime la orden a cambiar de estado<br><br>2-El chef una vez que esté listo una orden, oprime un botón que cambia el estado de “preparando” a ”listo” |
| **Postcondición** | Cambio de estado con éxito |
| **Flujo alternativo** | 2.1 El chef cambia de estado “listo” a “preparando” |
| **Excepciones** | No contempla |

---

| | |
|---|---|
| **Caso de uso** | CU008 Mostrar orden y sus estado |
| **Actores** | Caja, Chef, Administrador de datos |
| **Referencias** | **RF06** |
| **Descripción** | Muestra las ordenes y estados de cada una, para que la caja pueda cobrar y el chef ver lo que están pidiendo los clientes |
| **Precondición** | |
| **Flujo principal** | 1- El sistema llama a CU016<br>2- El sistema muestra todas las ordenes en estado preparando y las listas con máximo 5 minutos de ya estar listas |
| **Postcondición** | No contempla |
| **Flujo alternativo** | No contempla |
| **Excepciones** | No contempla |

---

| | |
|---|---|
| **Caso de uso** | CU009 Agregar mozo |
| **Actores** | Encargado, Administrador de datos |
| **Referencias** | **RF12** |
| **Descripción** | Permite al Encargado o Gerente agregar mozos |
| **Precondición** | Haber iniciado sesión en el sistema<br><br>Tener los permisos necesarios para agregar un nuevo mozo |
| **Flujo principal** | 1- El sistema muestra la interfaz de Encargado<br><br>2- El personal autorizado selecciona la opción Agregar mozo<br><br>3- El sistema muestra un formulario vacio para información personal sobre la persona a registrar con la siguiente información:<br><br>• Nombre<br>• Apellido<br>• Numero de contacto<br>• Correo electrónico<br>• DNI<br><br>4- El encargado confirma la entrada de datos<br><br>5- El sistema verifica si el numero de contacto, DNI y el correo no estén registrados<br><br>6- El sistema llama al caso de uso CU005 |
| **Postcondición** | |
| **Flujo alternativo** | Los datos ya están registrados:<br><br>5.1- El sistema muestra un mensaje de que los datos ya están en sistema<br><br>5.2 – El sistema vuelve al paso 2 |
| **Excepciones** | No contempla |

---

| | |
|---|---|
| **Caso de uso** | CU010 Quitar mozo |
| **Actores** | Encargado, Administrador de datos |
| **Referencias** | **RF14** |
| **Descripción** | Permite al Encargado o Gerente quitar mozos |
| **Precondición** | Haber iniciado sesión en el sistema<br>Tener los permisos necesarios para quitar un mozo |
| **Flujo principal** | 1- El sistema muestra la interfaz de Encargado<br>2- El personal autorizado selecciona la opción Quitar mozo<br>3- El sistema muestra todos los mozos registrados en el local<br>4- El personal selecciona el mozo a quitar<br>5- El sistema muestra un mensaje de para confirmar<br>6- El personal confirma<br>7- El sistema quita el mozo del sistema |
| **Postcondición** | Quitar el mozo seleccionado con éxito del sistema |
| **Flujo alternativo** | 5.1- Cancela acción en el mensaje para la confirmación<br>5.2- El sistema vuelve al paso 1 |
| **Excepciones** | No contempla |

---

| | |
|---|---|
| **Caso de uso** | CU011 Agregar mesa |
| **Actores** | Encargado, Administrador de datos |
| **Referencias** | **RF15, RF16** |
| **Descripción** | Permite al Encargado o Gerente agregar mesas |
| **Precondición** | Haber iniciado sesión en el sistema<br>Tener los permisos necesarios para agregar una nueva mesa |
| **Flujo principal** | 1- El sistema muestra la interfaz de Encargado<br>2- El personal autorizado selecciona la opción Agregar mesa<br>3- El sistema pide un numero para la mesa<br>4- El sistema verifica que el numero de la mesa no exista<br>5- El sistema genera un código para esa mesa<br>6- El sistema registra en el sistema la mesa con el numero y su código<br>7- El sistema muestra el código de la mesa junto con su numero |
| **Postcondición** | Mesa añadida con éxito |
| **Flujo alternativo** | El número de la mesa ya está registrado:<br>4.1- El sistema muestra un mensaje de que el número de la mesa ya están en sistema<br>4.2 – El sistema vuelve al paso 3 |
| **Excepciones** | No contempla |

---

| | |
|---|---|
| **Caso de uso** | CU012 Quitar mesa |
| **Actores** | Encargado, Administrador de datos |
| **Referencias** | **RF17** |
| **Descripción** | Permite al Encargado o Gerente quitar mesas |
| **Precondición** | Haber iniciado sesión en el sistema<br>Tener los permisos necesarios para quitar una mesa |
| **Flujo principal** | 1- El sistema muestra la interfaz de Encargado<br>2- El personal autorizado selecciona la opción Quitar mesa<br>3- El sistema pide el código de la mesa<br>4- El sistema verifica que el numero de la mesa exista<br>5- El sistema borra dicha mesa del sistema<br>7- El sistema muestra un mensaje de que se quito la mesa con éxito |
| **Postcondición** | Quitar mesa con éxito |
| **Flujo alternativo** | El código de la mesa no está registrado:<br>4.1- El sistema muestra un mensaje de que el código de mesa no existe<br>4.2 – El sistema vuelve al paso 3 |
| **Excepciones** | No contempla |

---

| | |
|---|---|
| **Caso de uso** | CU013 Agregar Ítem |
| **Actores** | Encargado, Administrador de datos |
| **Referencias** | **RF20** |
| **Descripción** | Permite al Encargado o Gerente agregar Ítem al menú |
| **Precondición** | Haber iniciado sesión en el sistema<br><br>Tener los permisos necesarios para agregar un nuevo ítem al menú |
| **Flujo principal** | 1- El sistema muestra la interfaz de Encargado<br><br>2- El personal autorizado selecciona la opción Agregar Ítem<br><br>3- E l personal autorizado selecciona a la sección que pertenecería<br><br>3- El sistema muestra un formulario vacio para información sobre el ítem a registrar con la siguiente información:<br><br>• Nombre/Titulo<br>• Breve descripción<br>• Precio<br><br>4- El encargado confirma la entrada de datos<br><br>5- El sistema genera un código para ese ítem<br><br>6- El sistema guarda el ítem en el sistema<br><br>6- El sistema muestra un mensaje de que se agrego con éxito el ítem |
| **Postcondición** | Ítem agregado con éxito |
| **Flujo alternativo** | No contempla |
| **Excepciones** | No contempla |

---

| | |
|---|---|
| **Caso de uso** | CU014 Quitar Ítem |
| **Actores** | Encargado, Administrador de datos |
| **Referencias** | **RF21** |
| **Descripción** | Permite al Encargado o Gerente quitar Ítem del menú |
| **Precondición** | Haber iniciado sesión en el sistema<br><br>Tener los permisos necesarios para quitar un ítem del menú |
| **Flujo principal** | 1- El sistema muestra la interfaz de Encargado<br><br>2- El personal autorizado selecciona la opción Quitar Ítem<br><br>3- E l personal autorizado selecciona a la sección que pertenecería<br><br>3- El sistema muestra un formulario vacio para información sobre el ítem a eliminar con la siguiente información:<br><br>• Código ID del ítem<br><br>4- El encargado confirma la entrada de datos<br><br>6- El sistema elimina el ítem en el sistema<br><br>6- El sistema muestra un mensaje de que se elimino con éxito el ítem |
| **Postcondición** | Ítem eliminado con éxito |
| **Flujo alternativo** | No contempla |
| **Excepciones** | No contempla |

---

| | |
|---|---|
| **Caso de uso** | CU015 Generar informes |
| **Actores** | Encargado, Administrador de datos |
| **Referencias** | **RF18** |
| **Descripción** | Permite al Gerente o Encargado generar informes |
| **Precondición** | Haber iniciado sesión en el sistema<br>Tener los permisos necesarios para generar los informes |
| **Flujo principal** | 1- El sistema muestra la interfaz de Encargado/ Gerente<br>2- El personal autorizado selecciona la opción informes<br>3- El sistema muestra los filtros/informes que se pueden hacer<br>4- El personal selecciona una opción<br>5-El sistema hace la búsqueda<br>6- El sistema muestra los resultados |
| **Postcondición** | No contempla |
| **Flujo alternativo** | No contempla |
| **Excepciones** | No contempla |

---

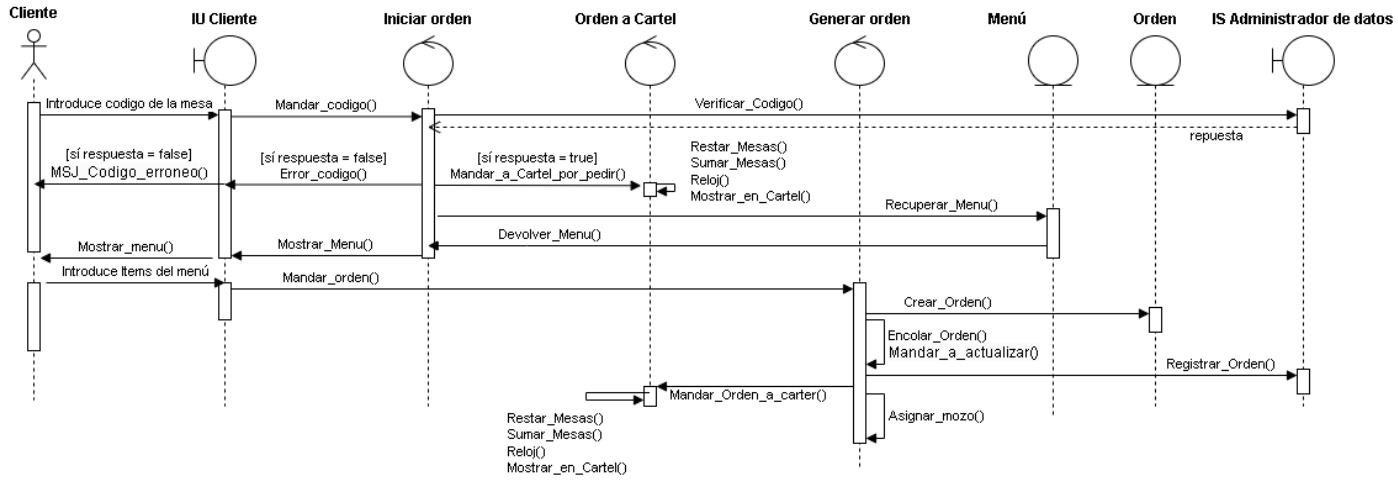
| | |
|---|---|
| **Caso de uso** | CU016 Consultar orden |
| **Actores** | Chef, Caja, Mozo, Administrador de datos |
| **Referencias** | **RF6, RF10** |
| **Descripción** | Consulta Datos sobre las ordenes |
| **Precondición** | |
| **Flujo principal** | 1- El caso de uso es llamado por CU006 o por CU008<br>2- El sistema hace una consulta a la base de datos sobre las ordenes<br>3- El sistema devuelve los resultados |
| **Postcondición** | Devolver los resultados con éxito |
| **Flujo alternativo** | No contempla |
| **Excepciones** | No contempla |

---

ETAPA DE ANALISIS: DIAGRAMAS DE SECUENCIA

## CLIENTE

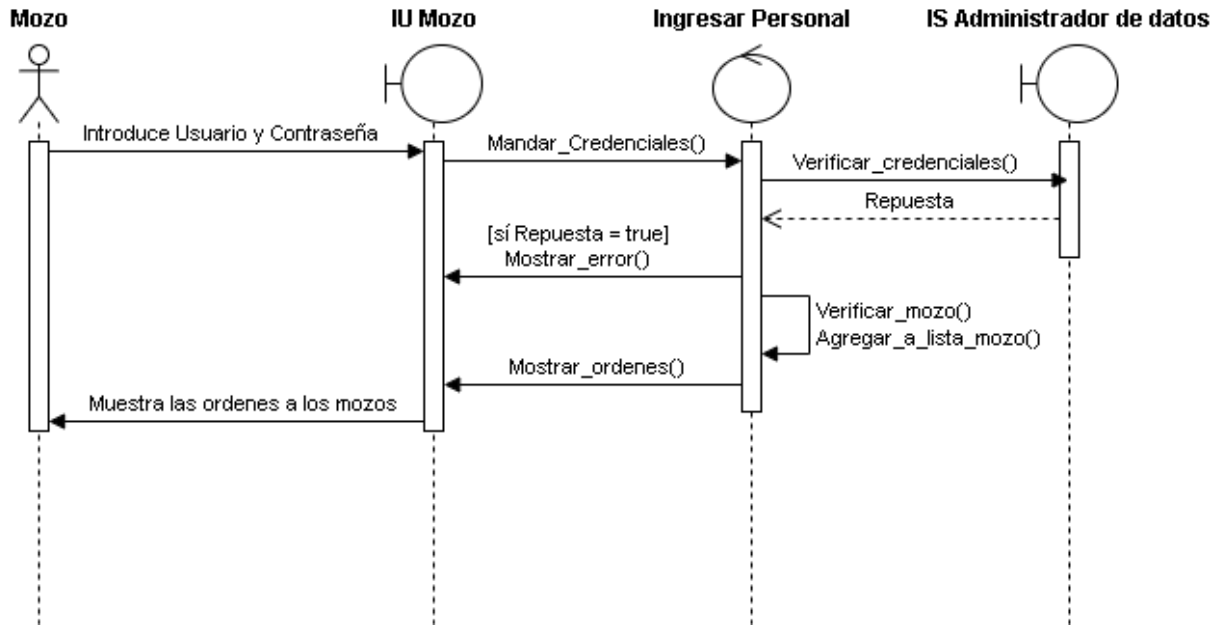
Orden:



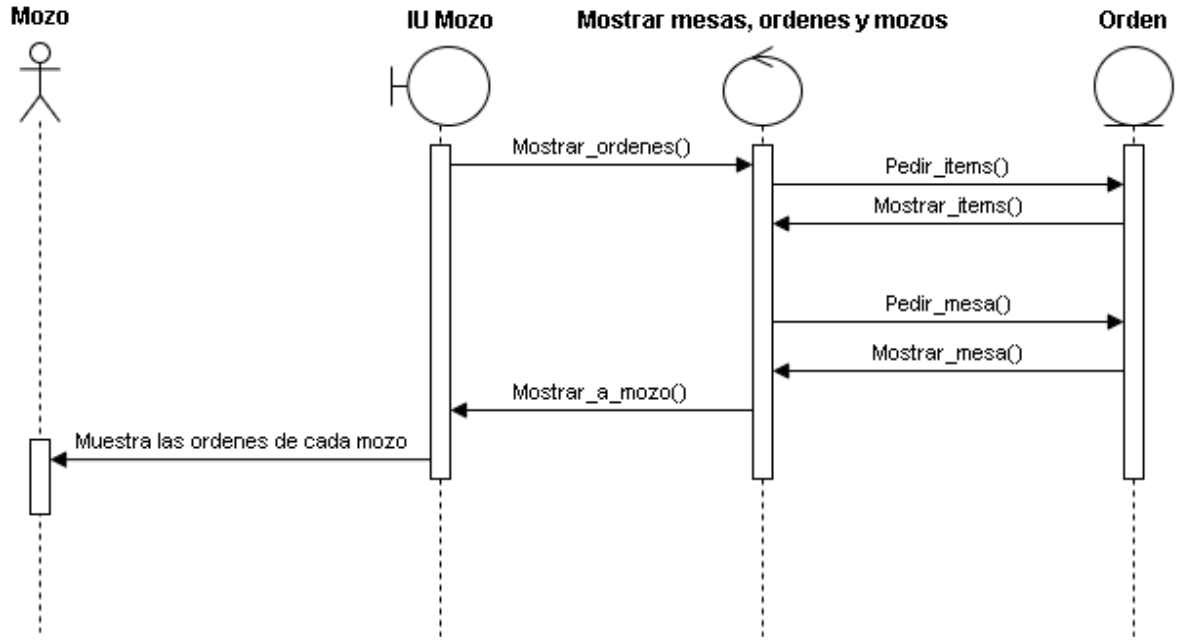
---

**MOZO**

Ingresar:



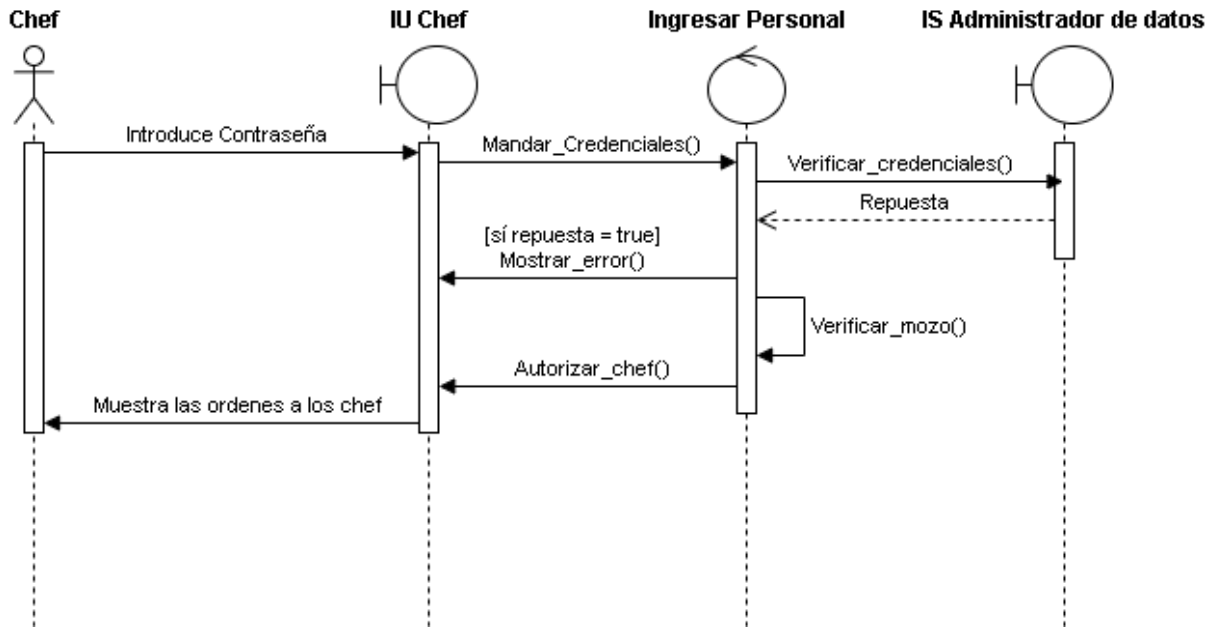

**Mostrar mesas:**



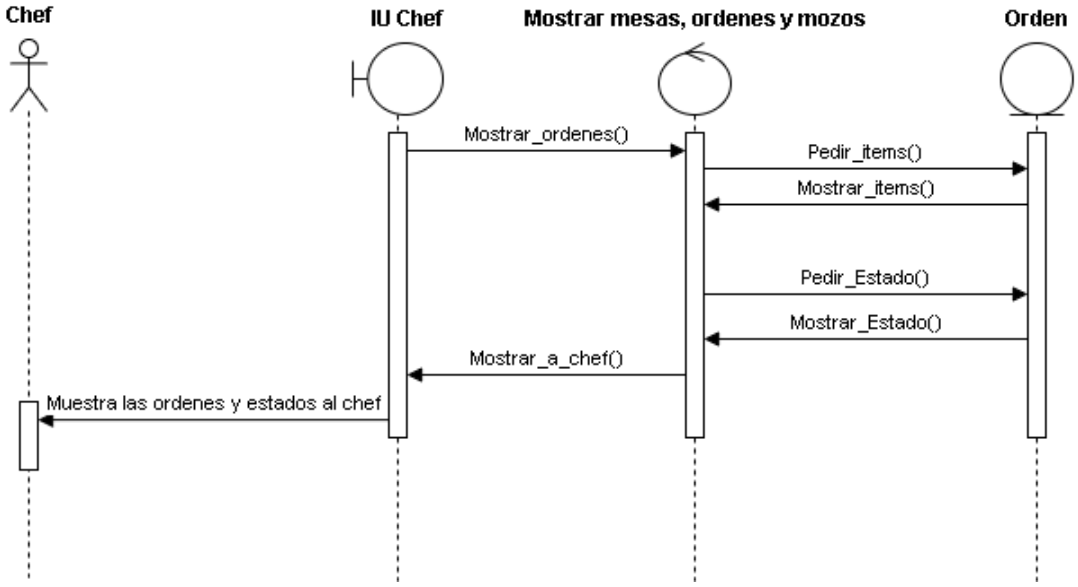
---

## CHEF

Ingresar:

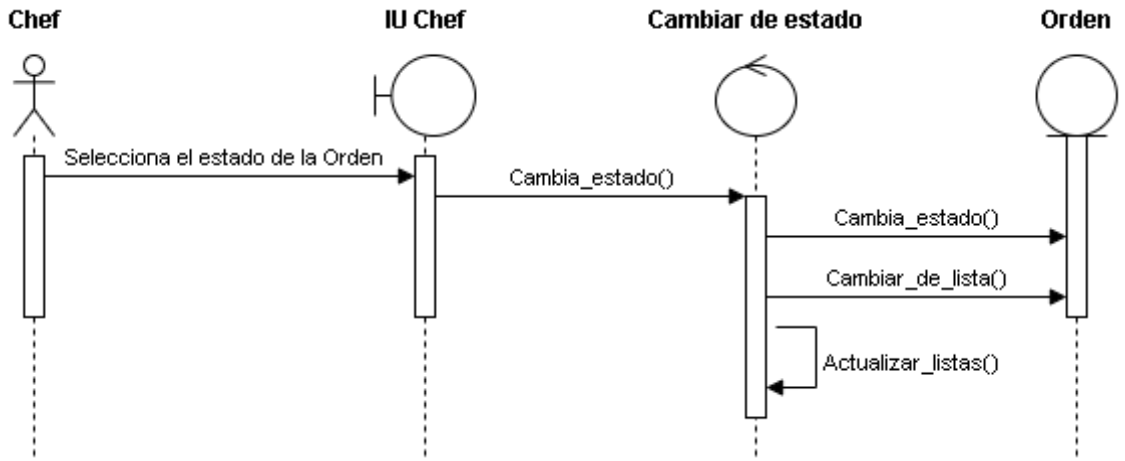



## Mostrar mesas:



---

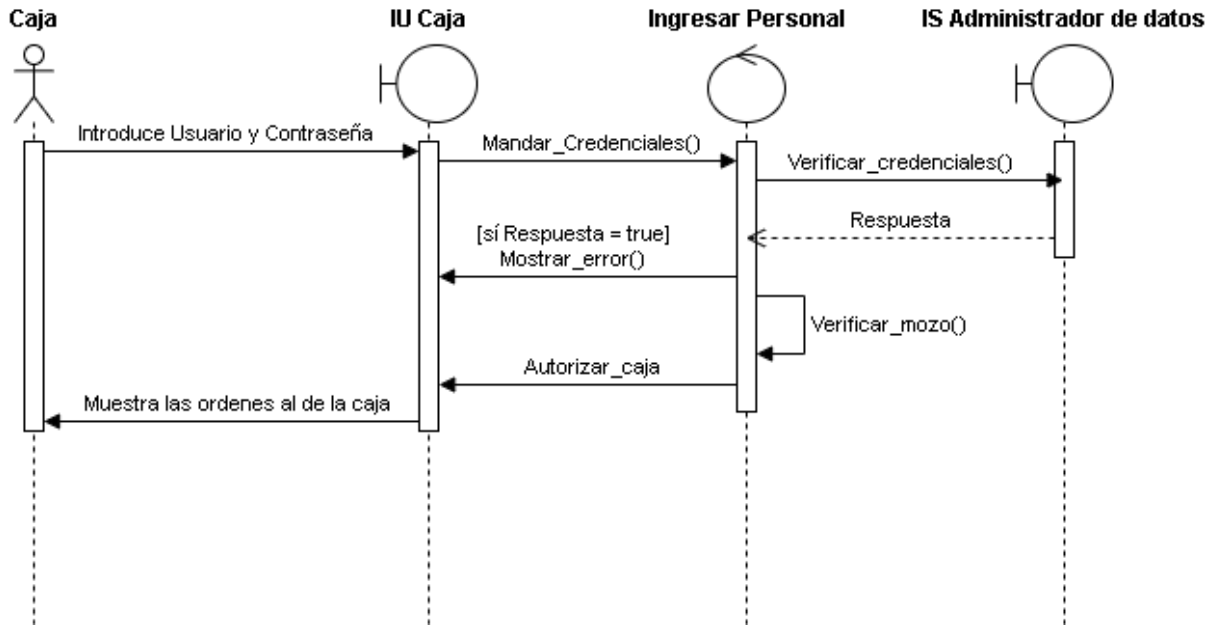
**Cambiar estado:**



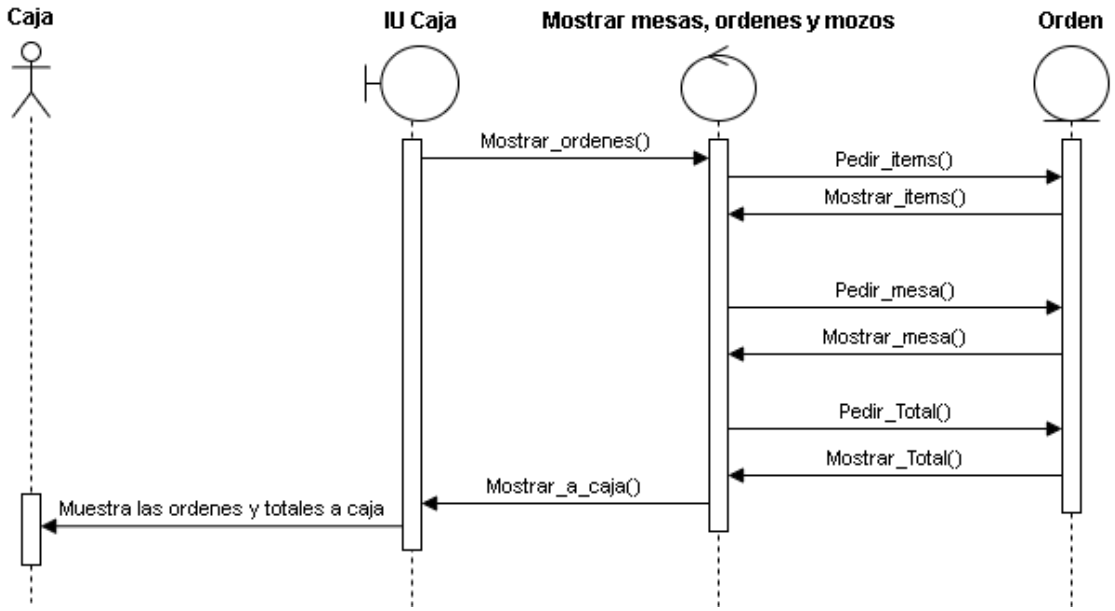
---

**CAJA**

Ingresar:



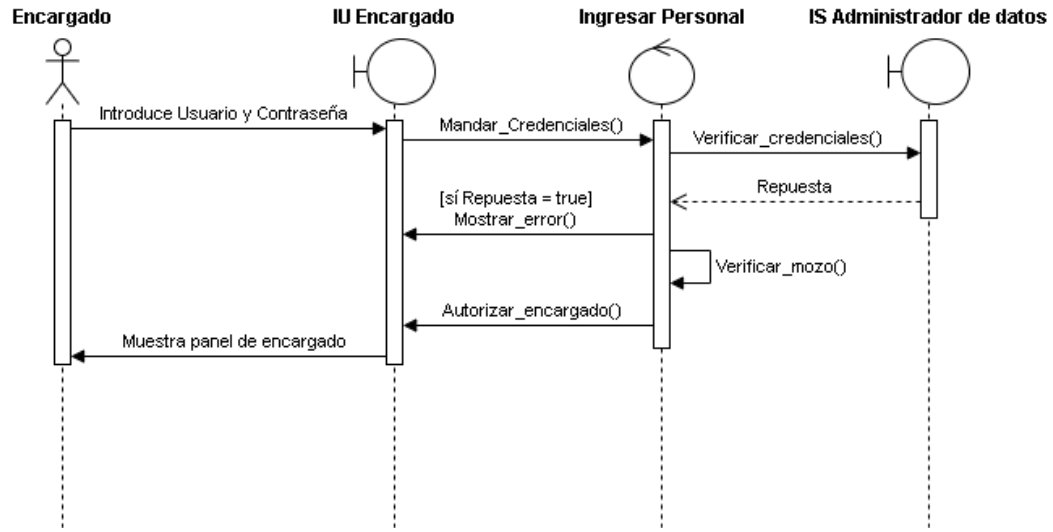

Mostrar mesas:



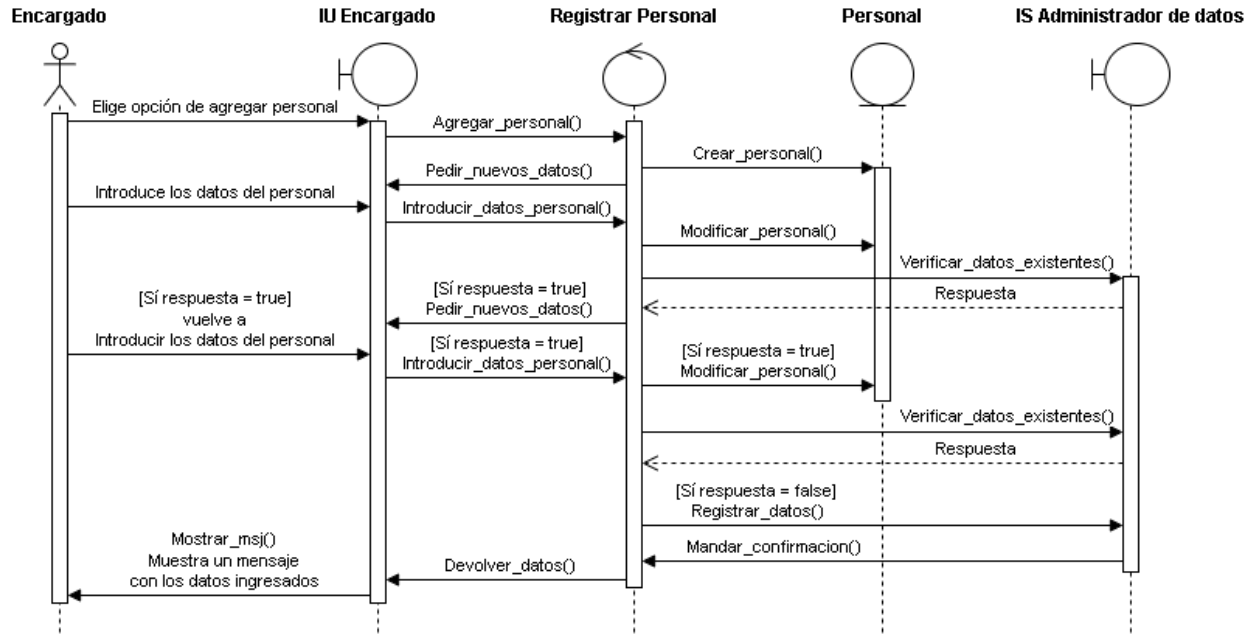
---

**ENCARGADO**

Ingresar:

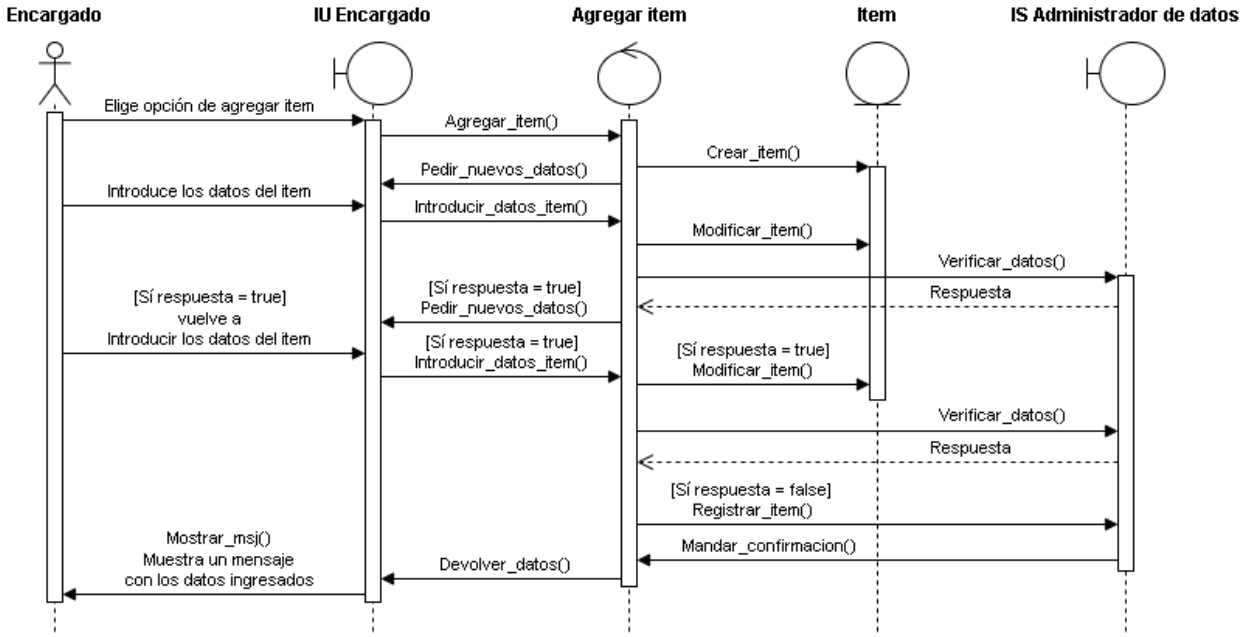



Agregar personal:

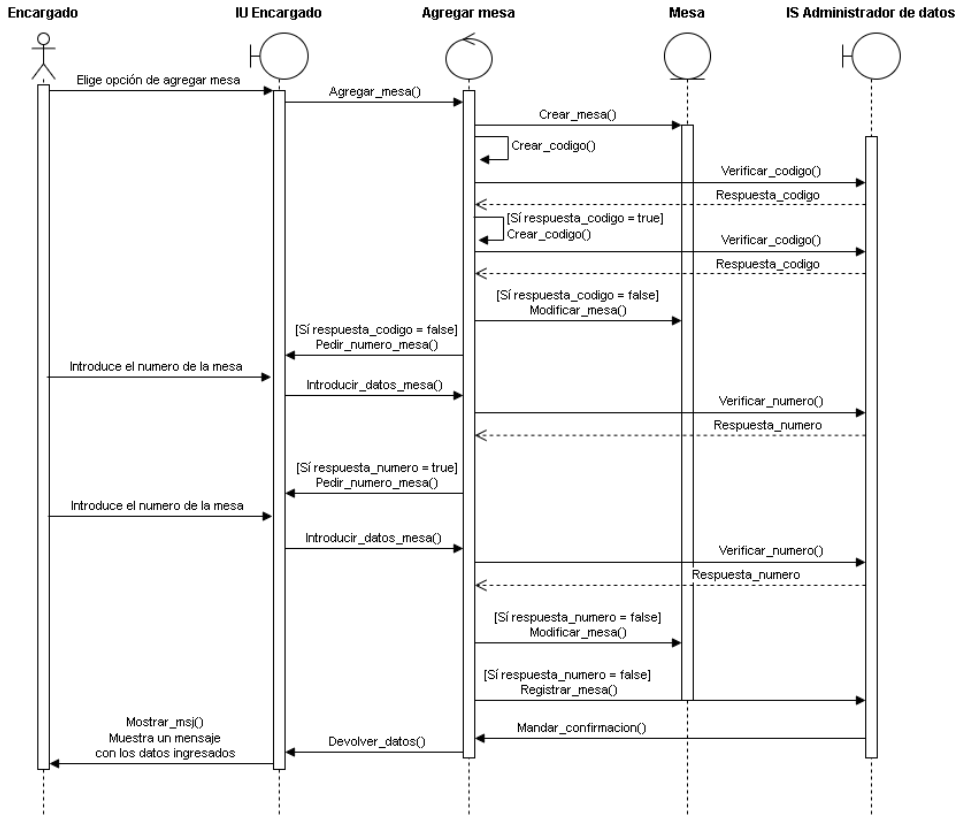


---

Agregar Ítem:

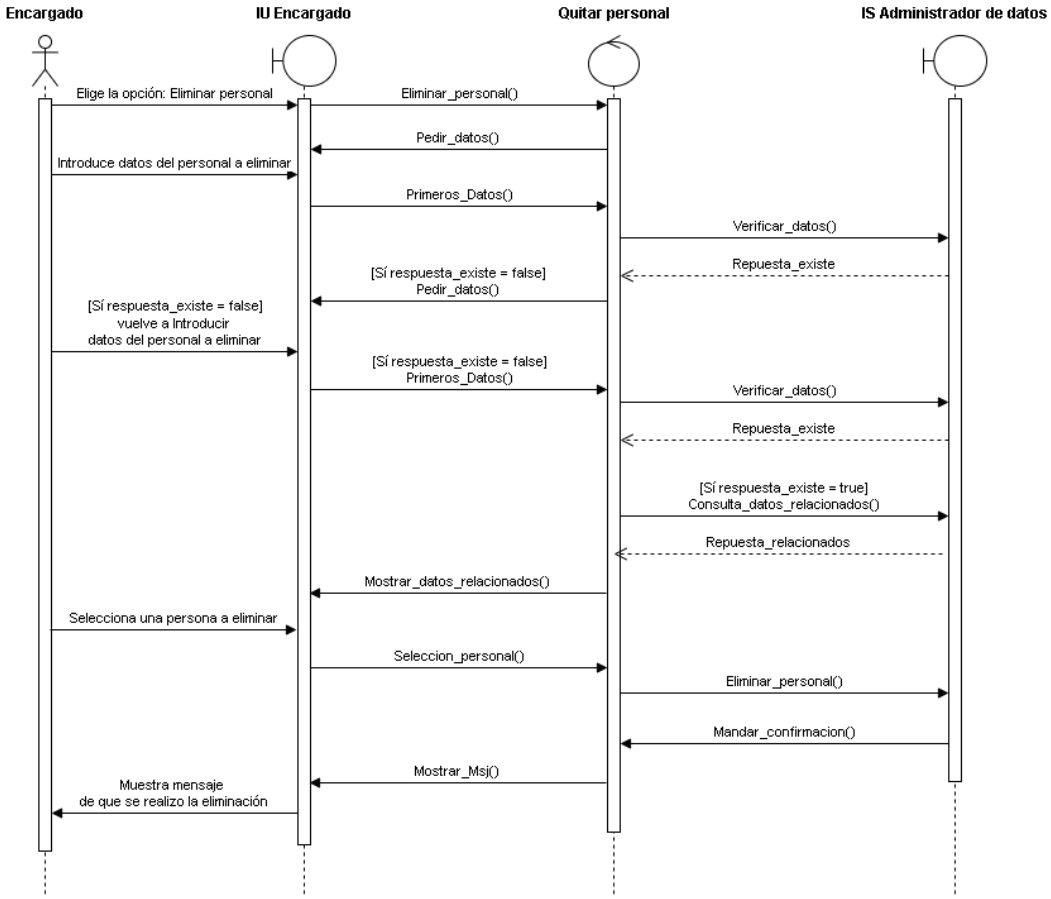



Agregar mesa:



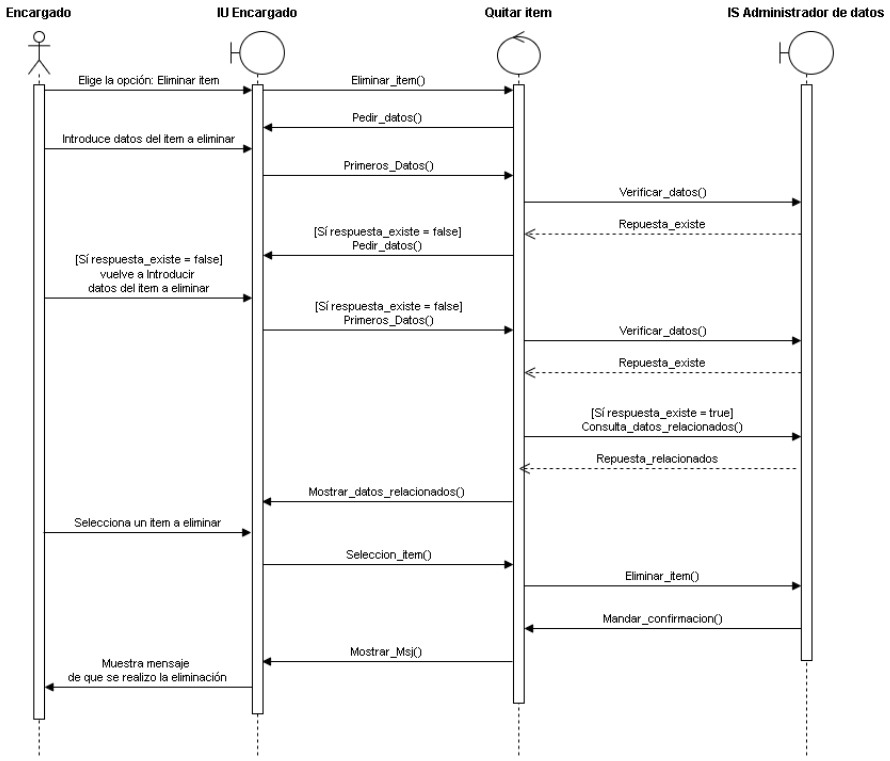
---

Quitar personal:

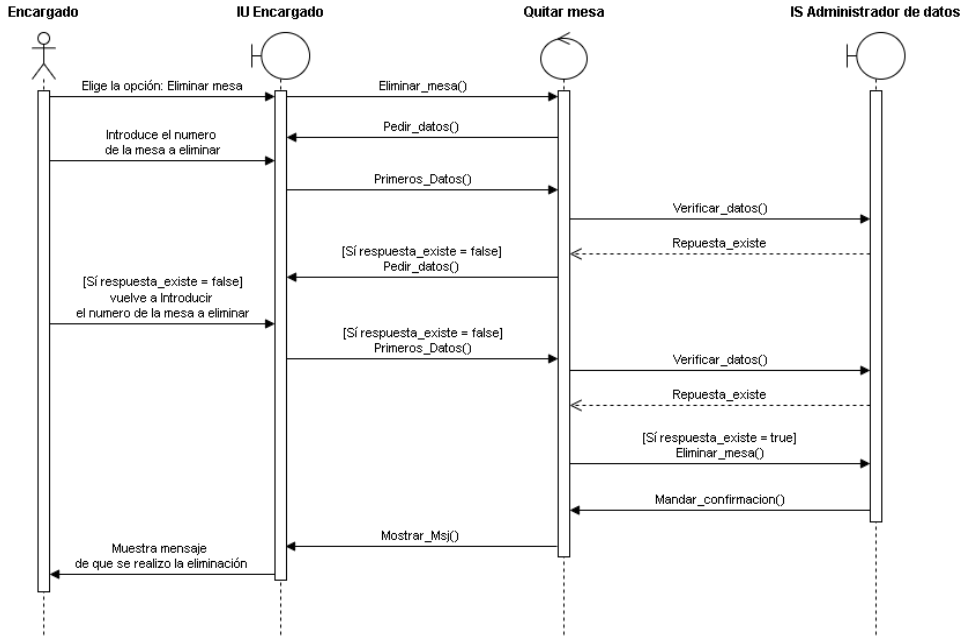


---

Quitar Ítem:




Quitar mesa:

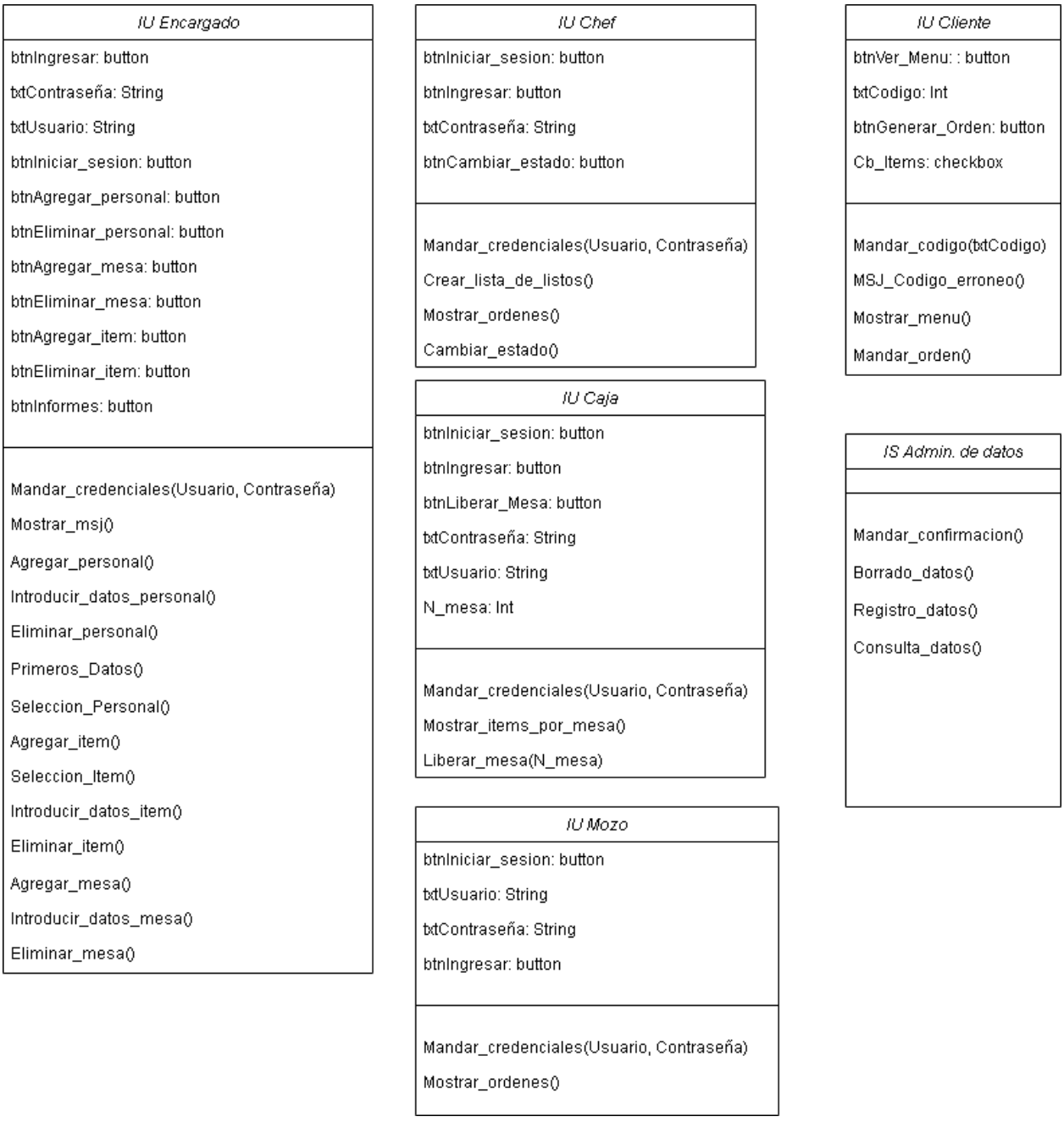


---

# ETAPA DE DISEÑO

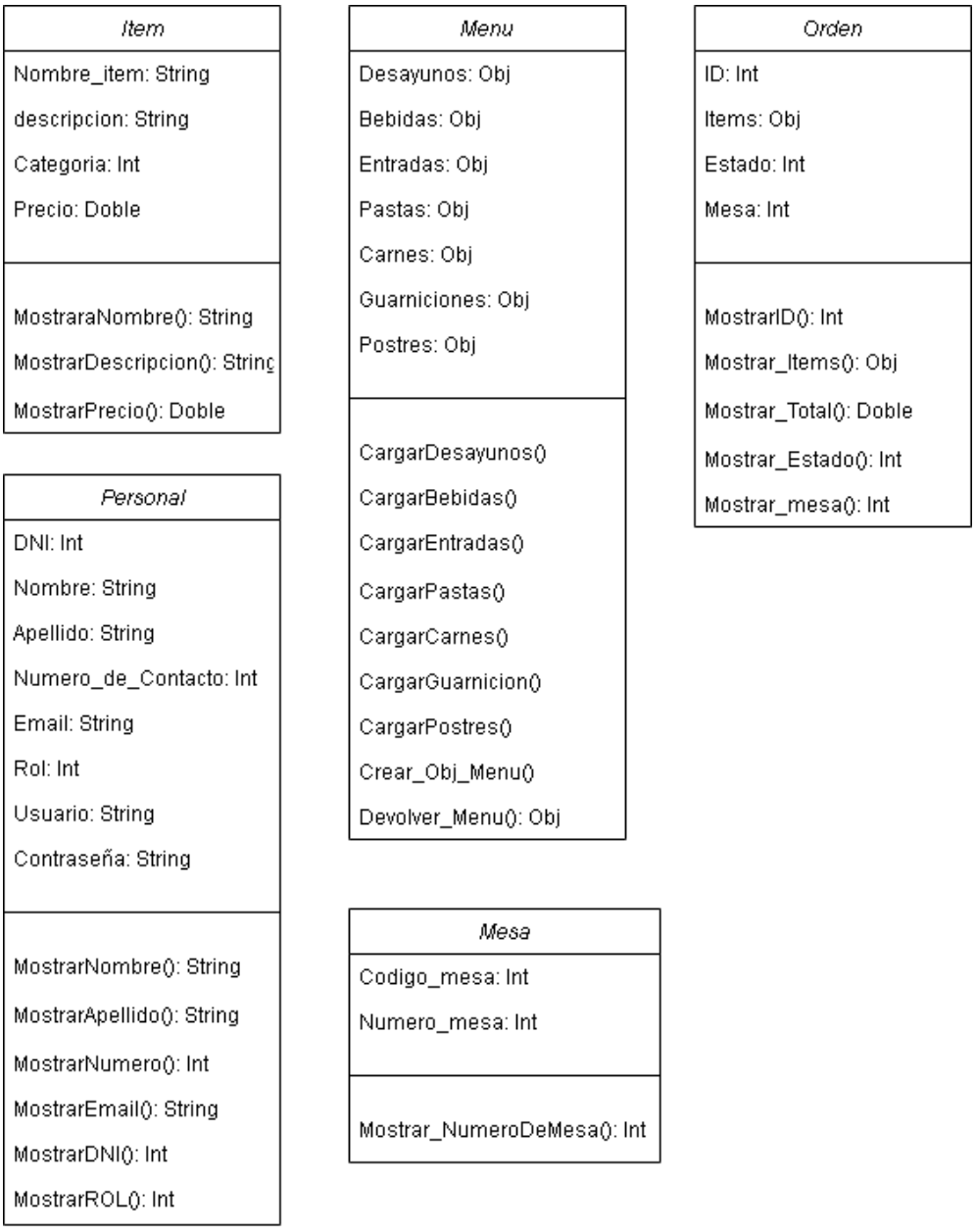
## CLASES DE DISEÑO:

## CLASES DE INTERFACES



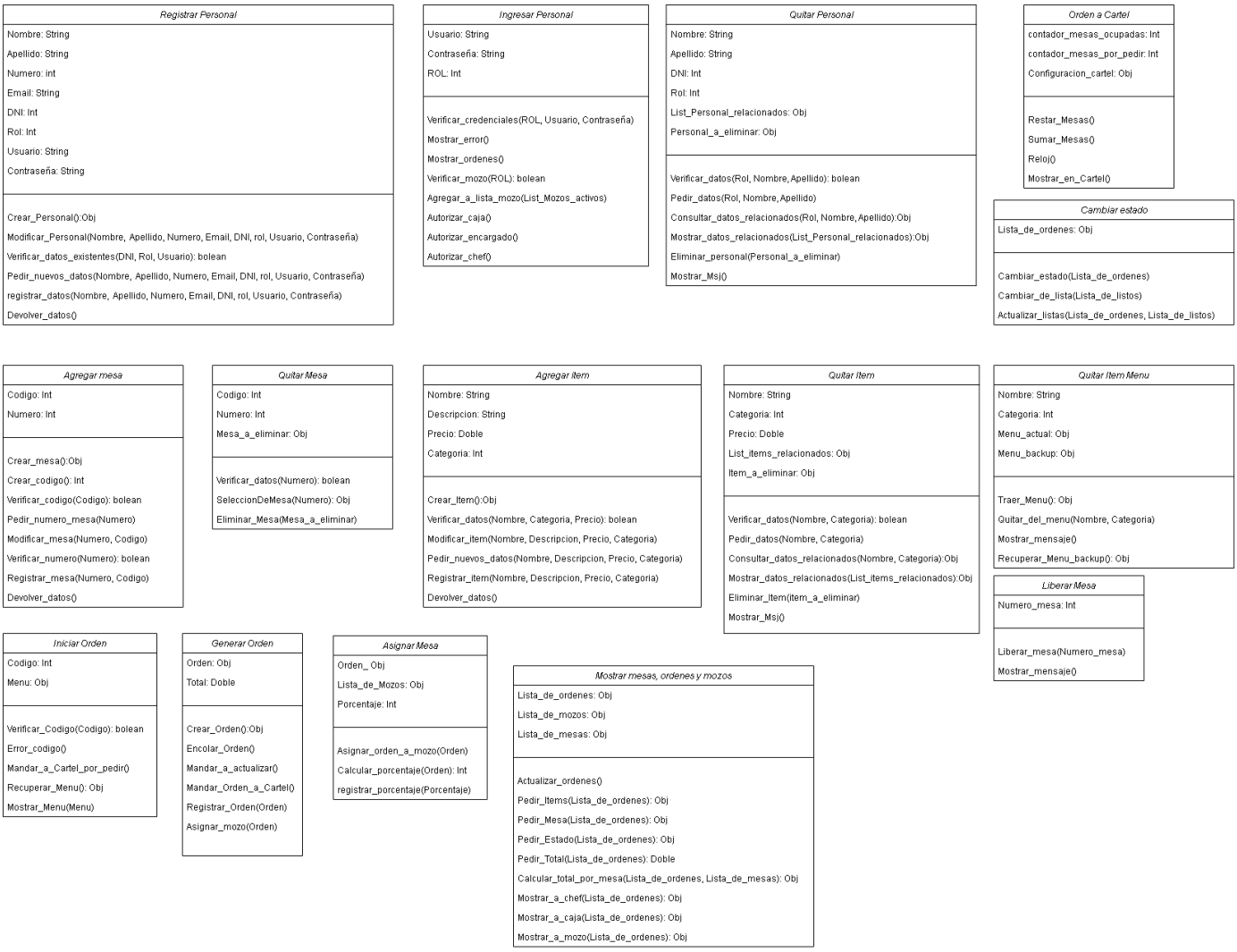
---

# CLASES DE ENTIDADES



---

# CLASES DE CONTROL



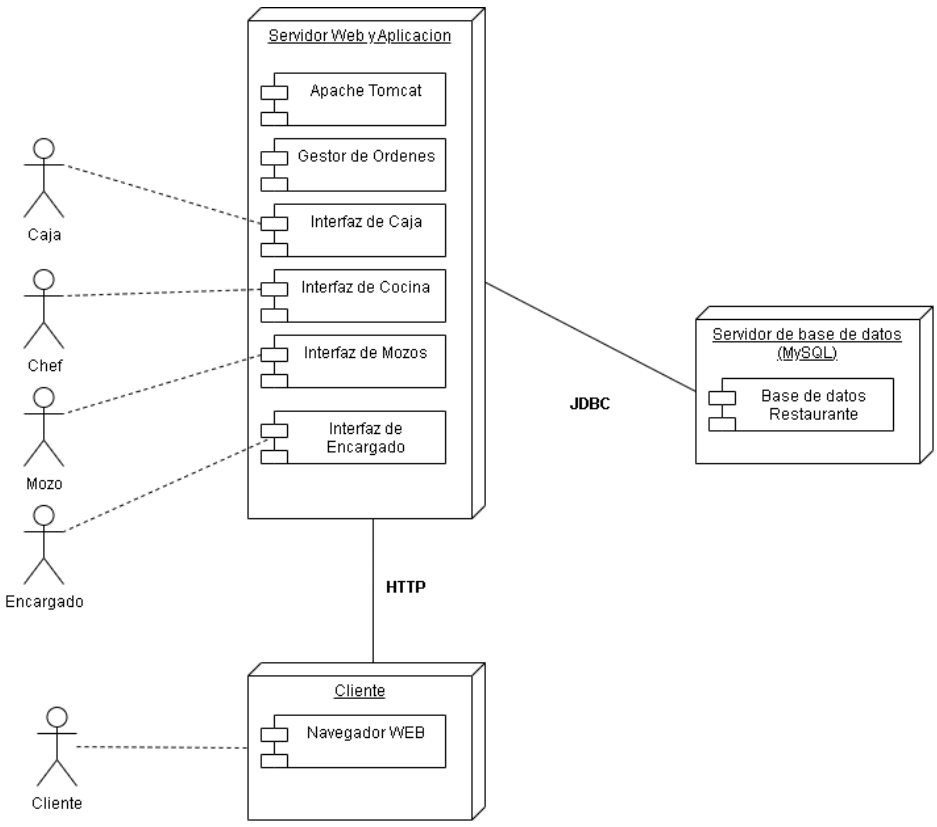
---

# ETAPA DE IMPLEMENTACION

## REQUERIMIENTOS:

| Requerimiento No Funcional | Descripción |
|---|---|
| **RNF001** | -El lenguaje de programación será java. |
| **RNF002** | - El sistema debe contar con una base de datos MySQL. |
| **RNF 003** | - El sistema debe utilizar XAMPP como entorno de desarrollo web de código abierto. |
| **RNF 005** | - Debe estar instalado en el propio restaurante |

## DIAGRAMA DE DESPLIEGUE:



---

# ETAPA DE PRUEBAS

## Plan de Pruebas

| Caso de uso | Código Prueba | Tipo de prueba | Técnica propuesta | Observaciones |
|---|---|---|---|---|
| **Iniciar Orden** | IO_CC | Componente | Partición de equivalencia - Frontera | Aplicación de caja negra en la clase: Iniciar Orden |
| **Mostrar Menú** | MM_A | Aceptación | Aceptación del Cliente que pidió el software | El Cliente prueba con el proveedor, para la aceptación del menú con respecto a los requisitos del cliente. |
| **Registrar personal** | RP_CC | Estados | Secuencia de Eventos | Verificación de todas las secuencia de eventos de la clase: Registrar Personal |
| **Ingresar personal** | IP_C | Componente | Cobertura | Aplicación de caja blanca en la clase: Ingresar Personal |
| **Eliminar personal** | EP_E | Estados | Tabla de Estados | Verificación de todos los estados de la clase: Eliminar Personal |
| **Generar orden** | GO_IG | Integración | Integración incremental | Integración de funcionalidades de la clase incrementalmente |
| **Orden (la entidad)** | O_IR | Interacción | Matriz CRUD | Aplicación de la matriz CRUD hacia el Objeto: Orden |

---

ESTRATEGIAS

| Código Prueba | Estrategia |
|---|---|
| **IO_CC** | Deberían estar probados el 100% de los casos de prueba y pasados con éxitos, y deben ser de manera manual |
| **MM_A** | Debería de cubrir el 70% de los requisitos descriptos por el cliente, hecho de forma manual por el cliente |
| **RP_CC** | Deberían estar probados el 80% de los Estados y pasados con éxitos, y deben ser de manera manual |
| **IP_C** | Al menos el 90% de las pruebas deberían ser exitosas, siendo en su totalidad de forma manual. Cada caso de uso será probado para su flujo normal y para un flujo alternativo. |
| **EP_E** | Deberían estar probados el 50% de los Estados y pasados con éxitos, y deben ser de manera manual |
| **GO_IG** | Deberían estar probados el 80% de las funciones del componente y pasados con éxitos, y deben ser de manera manual. |
| **O_IR** | Deberían estar probados el 100% de las interacciones y ser pasadas con éxitos, y deben ser de manera manual. |

---

# Casos de Pruebas

Caso de prueba IO_CC

| Requerimiento | Descripción |
|---|---|
| RNF008 | Cada mesa contara con un Identificador |
| RF001 | El sistema debe identificar cada mesa mediante el ingreso de un código (ID) |

***Al ser una prueba de partición de equivalencia, se tuvo que dividir los posibles valores de entrada en clases de equivalencias, por lo que terminamos teniendo las siguientes clases de equivalencias:***

**CLSES DE EQUIVALENCIAS VALIDAS**

| CEV1 |
|---|
| **100 000 ≤ e < 1 000 000** |

**CLSES DE EQUIVALENCIAS INVÁLIDAS**

| CEI1 | CEI2 |
|---|---|
| **100 000 > e** | **e ≥ 1 000 000** |

| Datos de entrada | Comportamiento esperado | Mensaje emitido por el sistema |
|---|---|---|
| -1000 | **No iniciar la Orden** | **Código erróneo** |
| 999 | **No iniciar la Orden** | **Código erróneo** |
| 100 000 | **Inicia la Orden** | **Bienvenido** |
| 999 999 | **Inicia la Orden** | **Bienvenido** |
| 1 000 000 | **No iniciar la Orden** | **Código erróneo** |
| 1 000 001 | **No iniciar la Orden** | **Código erróneo** |

---

Caso de prueba RP_CC

| Requerimiento | Descripción |
|---|---|
| RF012 | El sistema debe permitir al gerente agregar un nuevo mozo |
| RF013 | Al agregar un nuevo mozo, este debe poder ingresar su usuario y contraseña |

***Identificación de eventos:***

**1) Inicio:** El encargado accede al Sistema para registrar un nuevo personal

**2) Creación del objeto "Personal":** El sistema crea el objeto para guardar información sobre el personal

**3) Solicitud de datos:** El sistema pide los datos del personal a ingresar

**4) Modificación del objeto "personal" con los datos proporcionados**: El sistema guarda los datos proporcionados en el objeto "personal"

**5) Verificación de Datos:** El sistema verifica si los datos ya existen en la base de datos

- Si los datos son correctos, el sistema procede al registro
- Si los datos son incorrectos, el sistema vuelve a pedir los datos

**6) Registro de datos:** El sistema registra los datos del personal en la base de datos

**7) Devolución de los datos:** El sistema devuelve los datos a IU Encargado

**Inicio:** Verificamos que se está en el estado inicio

**Creación del objeto "Personal":** Verificamos la creación de dicho objeto, con un script que muestre por consola los objetos Personal activos en el sistema.

**Solicitud de datos:** Simulamos el envío de datos

**Modificación del objeto "personal" con los datos proporcionados**: Verificamos que los datos se guardaron en objeto, mediante un script que muestre los datos guardados en el objeto en consola

**Verificación de Datos (Correctos):** Verificamos que los datos correctos paran la validación

**Registro de datos:** Verificamos que la tabla personal en la base de datos, tenga un registro con los datos introducidos en la prueba

**Devolución de los datos:** Verificamos que los datos cargados en la base de datos y los devueltos por el sistema son los mismos

---

***Para la secuencia con error, se incluye pasos adicionales:***

**Verificación de Datos (Incorrectos):** Verificamos que los datos incorrectos no pasan la validación

**Solicitud de datos:** Simulamos la corrección de datos y el reenvío de los mismos

**Verificación de Datos (Correctos):** Finalmente, verificamos que los datos corregidos pasan la validación y pasan al registro de los mismos.

---

Procedimientos de pruebas

| | | |
|---|---|---|
| **Caso de Prueba:** IO_CC | Procedimiento de prueba: IO_CC1 | |
| **Resumen:** | Devolver Objeto Menú | |
| **Precondiciones:** | n/a | |
| **Datos de entrada:** | Un numero de 6 dígitos | |
| **Pasos de ejecución:** | 1 | El sistema corrobora que está dentro de los rangos de números correctos |
| | 2 | El sistema verifica que el código existe |
| | 3 | El sistema Manda una orden al componente Orden a Cartel |
| | 4 | El sistema Pide el objeto menu |
| | 5 | Devuelve el objeto menu |
| **Datos de salidas:** | Devolución del Objeto menu | |
| **Observaciones:** | Este procedimiento se hace con el componente sin conexión a base de datos, esto se simula y verifica el código dentro de esa simulación de base de datos. | |

---

| | | |
|---|---|---|
| **Caso de Prueba:** RP_CC | Procedimiento de prueba: RP_CC1 | |
| **Resumen:** | Registro de personal con éxito | |
| **Precondiciones:** | El usuario tiene que haber iniciado sesión como Encargado, y haber dado a la opción Agregar Personal | |
| **Datos de entrada:** | Datos Personales de la persona a registrar | |
| **Pasos de ejecución:** | 1 | El Sistema Crea un objeto “Personal” |
| | 2 | El Sistema Pide datos al usuario sobre la persona a ingresar |
| | 3 | El Encargado introduce los datos |
| | 4 | El Sistema Modifica el objeto “Personal” con datos dados por el Encargado |
| | 5 | El sistema verifica la existencia de esos datos en la base de datos |
| | 6 | El sistema Registra el personal nuevo, Sí no existen los datos en la Base de Datos |
| | 7 | El sistema devuelve los datos ingresados |
| **Pasos de ejecución alternativa:** | 6.1 | En caso de que si existen esos datos, EL sistema pide de nuevo los datos |
| | 6.2 | El usuario vuelve a introducir los datos con corrección |
| | 6.3 | El Sistema verifica nuevamente dichos datos |
| **Datos de salidas:** | Los datos del personal ingresado | |
| **Observaciones:** | n/a | |

---

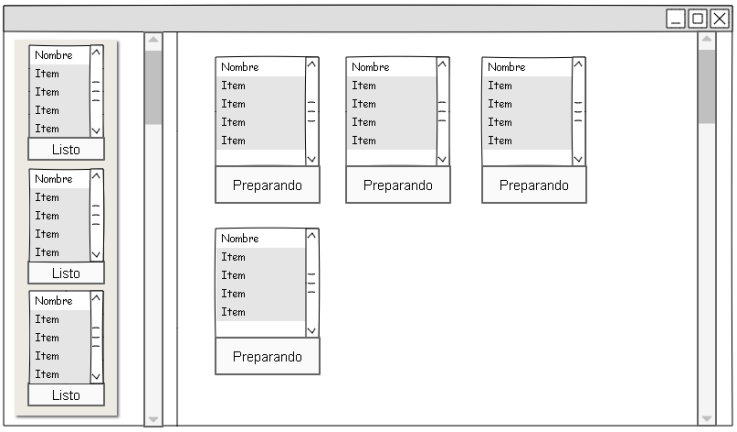
INTERFACES GRAFICAS

Cliente:

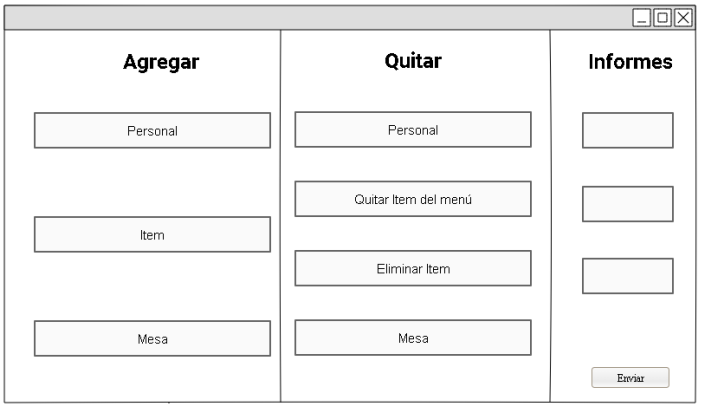

Mozos:

---

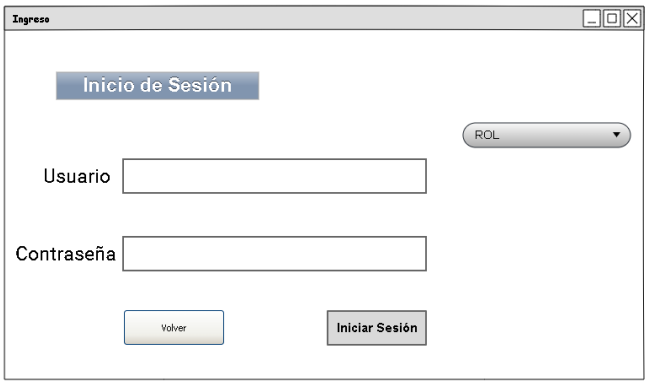
Chef:




Encargado:




Ingresar (Esta pantalla se comparte entre chef, mozos, caja y encargado para ingresar al sistema)



---

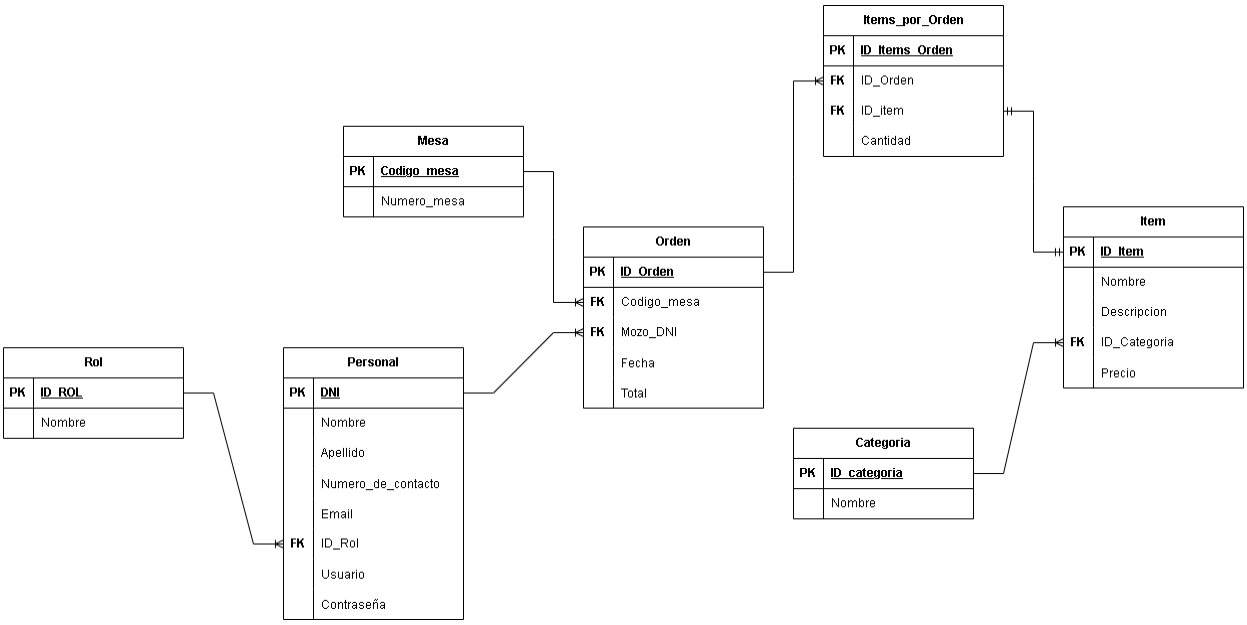
DEFINICION DE BASE DE DATOS PARA EL PROTOTIPO

Para el desarrollo del sistema de Gestión de las órdenes del restaurante, se requiere contar con una base de datos relacional. Tal cual como se especifica en los requerimientos no funcionales, en el prototipo se utiliza una base MySQL, que ofrece un buen rendimiento, flexibilidad y escalabilidad. Al considerar que se está trabajando con un prototipo, estas características son muy necesarias para avanzar de forma ordenada con la implementación del proyecto completo.

El uso de una base de datos es fundamental para lograr la persistencia de datos clave, tales como los ítems de cada orden, mesa a la que corresponde cada orden y su total a pagar, entre otros. Esto es esencial para mantener un historial completo sobre todas las ordenes y poder generar informes a futuro.

---

DIAGRAMA Entidad-Relación



---

## CREACION DE TABLAS

Tabla mesa:

```
CREATE TABLE Mesa
(
    Codigo_mesa     int primary key,
    numero_mesa     int
);
```

Tabla ROL:

```
CREATE TABLE ROL
(
    ID_ROL      int primary key,
    Nombre      varchar(45)
);
```

Tabla Categoria:

```
CREATE TABLE Categoria
(
    ID_categoria    int primary key,
    Nombre          varchar(45)
);
```

Tabla Personal:

```
CREATE TABLE Personal
(
    DNI                 int primary key,
    Nombre              varchar(25),
    Apellido            varchar(35),
    Numero_de_contacto  int,
    Email               varchar(50),
    ID_Rol              int,
    Usuario             varchar(45),
    Contraseña          varchar(45),
    foreign key(ID_Rol) references ROL(ID_ROL)
);
```

---

Tabla Orden:

```
CREATE TABLE Orden
(
    ID_Orden        int auto_increment primary key not null,
    Codigo_mesa     int,
    Mozo_DNI        int,
    Fecha           datetime,
    Total           decimal(9,2),
    foreign key(Codigo_mesa) references Mesa(Codigo_mesa),
    foreign key(Mozo_DNI) references Personal(DNI)
);
```

Tabla Item:

```
CREATE TABLE Item
(
    ID_Item         int primary key,
    Nombre          varchar(35),
    Descripcion     varchar(45),
    ID_Categoria    int,
    Precio          decimal(7,2),
    foreign key(ID_Categoria) references Categoria(ID_categoria)
);
```

Tabla Items_por_Orden:

```
CREATE TABLE Items_por_Orden
(
    ID_Item_Orden   int auto_increment primary key not null,
    ID_Orden        int,
    ID_Item         int,
    Cantidad        int,
    foreign key(ID_Orden) references Orden(ID_Orden),
    foreign key(ID_Item) references Item(ID_Item)
);
```

---

## INSERCION DE DATOS DE PRUEBA

Inserción de la tabla ROL:

```
insert into ROL(ID_ROL, Nombre)
values  (1 , 'Encargado'),
        (2 , 'Mozo'),
        (3 , 'Chef'),
        (4 , 'Caja');
```

Inserción de Ordenes (inserción en Orden y en Items_por_Orden):

```
DELIMITER //
create procedure Insertar_Orden(IN Codigo_MESA int, IN MOZO_DNI int, IN TOTAL Decimal(9,2), IN ID_ITEM int, IN CANTIDAD INT)
begin
insert into Orden (Codigo_mesa, Mozo_DNI, Fecha, Total)
values  (Codigo_MESA , MOZO_DNI , current_timestamp(), TOTAL);

Set @ultimoID = last_insert_id();

insert into Items_por_Orden (ID_Orden, ID_Item, Cantidad)
values  (@ultimoID , ID_ITEM , CANTIDAD);
END //
DELIMITER ;

CALL Insertar_Orden(802537, 2, 15000.00, 4, 1);
```

---

## CONSULTAS DE DATOS DE PRUEBA

```
select
Orden.ID_Orden as 'Identificador de orden',
Orden.Mozo_DNI as 'Mozo',
Orden.Fecha as 'Fecha',
Orden.Total as 'Total',
Mesa.Numero_mesa as 'Mesa numero',
Lista.Cantidad as 'Cantidad',
Item.Nombre as 'Nombre',
Item.Descripcion as 'Descripcion'
from
Orden,
Mesa,
Items_por_Orden as Lista,
Item

where
Orden.Codigo_mesa = Mesa.Codigo_mesa
and Orden.ID_Orden = Lista.ID_Orden
and Lista.ID_item = Item.ID_Item
;
```

## RESULTADO DE LA CONSULTA:

| Identificador de orden | Mozo | Fecha | Total | Mesa numero | Cantidad | Nombre | Descripcion |
|---|---|---|---|---|---|---|---|
| 4 | 16520122 | 2024-05-19 11:51:13 | 2500.00 | 9 | 1 | Ensalada | Lechuga y tomate |
| 5 | 22411714 | 2024-05-19 11:51:13 | 3000.00 | 5 | 2 | Helado | 1 Bocha de Frutilla |
| 6 | 33455213 | 2024-05-19 11:51:13 | 1600.00 | 1 | 1 | Cafe | Molido |
| 7 | 16520122 | 2024-05-19 11:51:13 | 3500.00 | 12 | 1 | Fideos | Fideos cinta |
| 8 | 22411714 | 2024-05-19 11:51:13 | 6000.00 | 7 | 1 | Pizza | Comun |
| 9 | 33455213 | 2024-05-19 11:51:13 | 2600.00 | 4 | 1 | Puré | De papa |
| 10 | 16520122 | 2024-05-19 11:51:13 | 15000.00 | 9 | 1 | Parrillada | Asado de tira, Morcilla, Chorizo, Corte de cerdo |
| 11 | 22411714 | 2024-05-19 11:51:13 | 5000.00 | 3 | 1 | Milanesa | De Carne de vaca |
| 12 | 33455213 | 2024-05-19 11:51:14 | 1200.00 | 10 | 1 | Empanada | De Carne |

## BORRADO DE LOS DATOS DE PRUEBA Y SU VERIFICACION

```
delete from Orden;
delete from Items_por_Orden;

select * from Orden;
select * from Items_por_Orden;
```

---

**DEFINICION DE COMUNICACION**

**Descripción del Sistema**

El sistema de un restaurante consta de una PC servidor que aloja la base de datos y el programa principal. Esta PC está conectada por cable a varias pantallas y a un cartel LED. Además, los clientes pueden hacer pedidos a través de una página web del restaurante.

## Requerimientos de Comunicación

Conectividad de Red:

**Local (LAN):** La PC servidor debe estar conectada por cable a las pantallas y al cartel LED, asegurando una conexión estable y de baja latencia.

**Externa (Internet):** La página web del restaurante debe estar accesible desde cualquier dispositivo con conexión al WIFI del restaurante para que los clientes puedan hacer pedidos.

Velocidad y Confiabilidad:

La comunicación debe ser rápida y confiable para actualizar el estado de las órdenes en tiempo real.

---

## Protocolos y Estándares de Comunicación

**Protocolo de Red Local (LAN):**

**Ethernet:** Para la conexión por cable entre el servidor, las pantallas internas y el cartel LED.

**TCP/IP:** Para asegurar la transmisión de datos confiable dentro de la red local.

**Protocolo de Internet:**

HTTP:es preferido para la comunicación entre la página web y el servidor

**Base de Datos:**

SQL: Para las interacciones con la base de datos, permitiendo consultas y actualizaciones eficientes.

## Entorno de Red e Infraestructura Física

**Cableado:**

**Categoría 6 (Cat6) o superior:** Para conexiones Ethernet, asegurando alta velocidad y baja latencia en la red local.

**Dispositivos de Red:**

**Switches Gigabit Ethernet:** Para conectar todos los dispositivos en la red local, asegurando suficiente ancho de banda.

**Servidores:**

Servidor dedicado o PC robusta: Con suficiente capacidad de procesamiento, memoria y almacenamiento para manejar la base de datos y la aplicación del restaurante.

---

**LINK DE GITHUB DE LA BASE DE DATOS:** *https://github.com/Giuliano-Kuhn/BD*

---

# DESARROLLO DEL SISTEMA UTILIZANDO JAVA

**Explicación del desarrollo en Java**

De acuerdo con los requerimientos no funcionales especificados, el sistema se debe desarrollar en Java. Se debe lograr un código estructurado, legible y modular, para facilitar la comprensión, el mantenimiento y la escalabilidad del proyecto.

Sin perder de vista que el proyecto debe contemplar el desarrollo del sistema completo, en esta instancia se va a trabajar con un prototipo que permita obtener un mínimo producto viable.

El proyecto o prototipo desarrollado debe contemplar necesariamente las siguientes características:

- Correcta utilización de sintaxis, tipos de datos, estructuras de control.
- Tratamiento y manejo de excepciones.
- Adecuada aplicación del encapsulamiento, herencia, polimorfismo y abstracción.
- Disponibilidad de un menú de selección.
- Empleo de estructuras condicionales y repetitivas.
- Declaración y creación de objetos en Java.
- Utilización de constructores para inicializar objetos.
- Uso de algoritmos de ordenación y búsqueda (opcional según el desarrollo).

Las imágenes que se presentaran reflejaran el desarrollo del código referido a:

Los casos de usos:

- *Iniciar Orden*
- *Generar Orden*

Diagrama de Secuencia:

Cliente*: Orden*

***Aclaración:***

- El proyecto está pensado para iniciarse desde el archivo llamado Main.java, porque es donde se inician las listas que se mantendrán en toda la ejecución guardando información como, los mozos activos y las ordenes con sus respectivas listas de ítems seleccionados por el cliente del restaurante.
- El arreglo que se llama mesas no existiría en el programa final, ya que se haría con consultas a una base de datos, al igual que la creación de los mozos activos, que sería añadidos a medida que se inician sesión en el programa.
- El método Mandar_a_cartel() esta vacio porque allí iría código para manipular un cartel led.

---

**Tratamiento y manejo de excepciones.**

Aquí se usa try para intentar redireccionar al usuario a otra pagina llamada index.jsp, en caso de algún problema, por ejemplo que no se encuentre el archivo index.jsp o se corte la conexión, etc; seria capturado por el catch con una excepción IOException que mostraría que ocurrió un error.

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)  //Sobreescribe el metodo diGet para manejar las solici
        throws ServletException, IOException {
      try {   //se hace un try para capturar algun error que pueda pasar.
            response.sendRedirect("index.jsp"); //Redirige la respuesta HTTP a la pagina index.jsp
        } catch (IOException ex) {
            processRequest(request, response);//Maneja la excepcion imprimiendo un mensaje de error
        }
}


    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>ERROR</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1> Ocurrió un error en mostrar la pagina inicial </h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

---

**Adecuada aplicación del encapsulamiento, herencia, polimorfismo y abstracción.**

**Encapsulamiento:**

El encapsulamiento surge aquí en la restricción de *prívate* ya que lo que hace esto, es que solo la clase *Personal* pueda acceder a esos atributos.

```
class Personal {
    private int DNI;
    private String Nombre;
    private String Apellido;
    private int Numero_de_Contacto;
    private String Email;
    private int ROL;
    private String Usuario;
    private String Contrasena;
```

**Herencia:**

Aquí la clase Mozo hereda de la clase Personal los atributos y métodos.

```
public class Mozo extends Personal{
    public int N_mesas;
    List<Orden> Mozo_Ordenes = new LinkedList<>();

    public Mozo(int DNI, String Nombre, String Apellido){
        super(DNI, Nombre, Apellido);
        N_mesas = 0;
    }
}
```

---

**Polimorfismo:**

Aquí el Polimorfismo se hace presente en los metodos doGet de los servlets, ya que es un método de la super clase HttpServlet, donde con @Override sobrescribo el comportamiento de dicho método, haciendo que en cada servlets el doGet haga distintas operaciones.

```
@WebServlet(name = "Iniciar_Orden", urlPatterns = {"/Iniciar_Orden"})
public class Iniciar_Orden extends HttpServlet {
    public Menu menuM; //Declara una instancia de la clase Menu
    public List menu;  //Declara una Lista para el menu
    private final int[] mesas = {635384,802537,282646,898767,556675,114345,685384,812537,262646,808767,666675,652238}; // Este arreglo
    private boolean verificacion_codigo;
    private int codigo;

    @Override   //Sobreescrime el metodo doGet
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String Codigo = request.getParameter("COD");    //se trae del formulario anterior el codigo(COD) que se introduzco y se lo asig
        codigo = Integer.parseInt(Codigo);              //Se pasa a int la varable Codigo para una comparacion futura
        verificacion_codigo = Verificar_Codigo();       //Se llama al metodo Verificar_Codigo() y el resultado se lo asigna a la variab
        if(verificacion_codigo){                        //Se consulta si el codigo es valido
            Recuperar_Menu();
            HttpSession misesion = request.getSession();//Se trae la sesion Actual
            misesion.setAttribute("Menu", menu);        //Guarda en la sesion el menu
            misesion.setAttribute("Nmesa",codigo);      //Guarda en la sesion el codigo de mesa
            Mostrar_Menu(request, response);            //llama a Mostrar_Menu()

        }
    }
```

---

**Abstracción:**

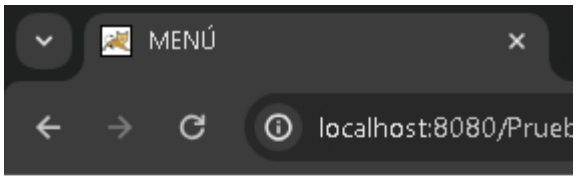
Aquí la abstracción se hace presencia en las listas, ya que se importan las librerías de List y LinkedList, haciendo que se pueda usar el método add(), abstrayéndonos el cómo funciona internamente y haciendo foco solo en la funcionalidad del método.

```
public class MEMO { //Esta clase se crea para mantener
        static List<Orden> List_Ordenes;
        static List<Mozo> Mozos;

    public static void iniciarlizarListas() {
        Mozo mozo1 = new Mozo(1,"Giuliano", "Kuhn");
        Mozo mozo2 = new Mozo(2,"Gladiz", "Ostertag");
        Mozo mozo3 = new Mozo(3,"Darwin", "Vila");
        Mozos.add(mozo1);
        Mozos.add(mozo2);
        Mozos.add(mozo3);
    }

    public static void CrearListas(){
        List_Ordenes = new LinkedList<>();
        Mozos = new LinkedList<>();
    }

    public void MostrarLISTA(){
       System.out.println(List_Ordenes);
    }
}
```

---

**Disponibilidad de un menú de selección.**

Aquí se puede observar un prototipo de  menú de selección del menú del restaurante, pudiendo seleccionar los ítem que se deseen consumir y luego enviar la orden al sistema.




# MENÚ

- ☐ **cafe ---------- 1200.0**
- ☐ **Té ---------- 1200.0**
- ☐ **Coca-cola ---------- 2000.0**
- ☐ **Vino ---------- 6000.0**
- ☐ **Empanada ---------- 1200.0**
- ☐ **Picada ---------- 8000.0**
- ☐ **Fideos ---------- 3500.0**
- ☐ **Pizza ---------- 8000.0**
- ☐ **Parrillada ---------- 10000.0**
- ☐ **Suprema ---------- 4500.0**
- ☐ **Ensalada ---------- 2500.0**
- ☐ **Papas ---------- 3500.0**
- ☐ **Helado ---------- 1500.0**
- ☐ **Bundin De Pan ---------- 2500.0**

Enviar

---

**Empleo de estructuras condicionales y repetitivas.**

Aquí se utilizan la estructura condicional *if* para verificar que el código ingresado por el cliente exista como código de alguna mesa.

Mientras que se usa un *for each* para iterar, en este caso, el arreglo que contiene los código de las mesas

```
public boolean Verificar_Codigo(){
  boolean verificacion_codigo = false;
  for(int elemento : mesas){   //for each para iterar en el arreglo de mesas
     if(elemento == codigo){  //compara para ver si existe el codigo en el arreglo de mesas
        verificacion_codigo = true;
        break;
     }
  }
   return verificacion_codigo;}//devuelve el valor de verificacion_codigo
```

**Declaración y creación de objetos en Java.**

Aquí se declara y crean 3 objetos de tipo Mozo (mozo1, mozo2, mozo3) que son luego añadidos a la lista Mozos

```
public static void iniciarlizarListas() {
    Mozo mozo1 = new Mozo(1,"Giuliano", "Kuhn");
    Mozo mozo2 = new Mozo(2,"Gladiz", "Ostertag");
    Mozo mozo3 = new Mozo(3,"Darwin", "Vila");
    Mozos.add(mozo1);
    Mozos.add(mozo2);
    Mozos.add(mozo3);
}
```

---

**Utilización de constructores para inicializar objetos.**

Aquí en la clase Mozo se utiliza un constructor con parámetros DNI, Nombre y Apellido, lo que inicializa N_mesas en 0, y como es una clase que hereda de Personal atributos, se llama al constructor de la clase Personal mediante super(DNI, Nombre, Apellido).

```
public class Mozo extends Personal{
    public int N_mesas;
    List<Orden> Mozo_Ordenes = new LinkedList<>();

    public Mozo(int DNI, String Nombre, String Apellido){
        super(DNI, Nombre, Apellido);
        N_mesas = 0;
    }
}
```

---

**Uso de algoritmos de ordenación y búsqueda (opcional según el desarrollo).**

En este proyecto se utiliza un algoritmo de búsqueda exaustiva, donde el objetivo es buscar los ítems de la lista MENU para poder guardarlos en T_Pedidos como objetos de tipo Item, y se lo busca en base al arreglo values que contiene los ítems seleccionados del cliente.

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //processRequest(request, response);
    String[] values = request.getParameterValues("items");              //Se trae
    List<List> MENU = (List) request.getSession().getAttribute("Menu"); //Se trae
    int N_mesa = (int) request.getSession().getAttribute("Nmesa");      //Se trae
    List<Item> T_Pedidos= new LinkedList<>();                           //Se inic
    //System.out.println(values.length);                                  //DEBUG
    //System.out.println(N_mesa);                                         //DEBUG
    if(values != null){                                                 //Pregunt
        for(List<Item> categorias : MENU){                              //Itera e
            for(Item item : categorias){                                //Itera l
                for(int i = 0; i < values.length; i++) {                //Itera l
                    if (item.nombre_Item.equals(values[i])) {           //Compara
                        T_Pedidos.add(item);                            //Para lu
                        //System.out.println(item.nombre_Item);           //DEBUG

                    }
                }
            }
        }
    Generar_Orden_BACK GOB = new Generar_Orden_BACK();                  //Declara
    GOB.Crear_Orden(T_Pedidos, N_mesa);                                 //llama a
    }
}
```

---

# CONEXIÓN DE LA BASE DE DATOS CON JAVA

El proyecto o prototipo desarrollado en Java debe contemplar necesariamente las siguientes características:

- Corrección de las observaciones realizadas en la entrega de las actividades anteriores.
- Selección del patrón de diseño y justificación.
- Persistencia y consulta de datos en una base de datos MySQL. Esta actividad incluye establecer conexiones, realizar consultas, actualizar registros y presentar resultados en la interfaz.
- Correcta aplicación de excepciones para la interacción con la base de datos MySQL.
- Inclusión pertinente de clases abstractas o interfaces.
- Utilización complementaria de arreglos y de la clase *ArrayList*.

---

**Selección del patrón de diseño y justificación.**

Un patrón de diseño de software proporciona una estructura predefinida que ayuda a resolver de manera eficiente los problemas que se presentan en el desarrollo de una aplicación.

Cada patrón de diseño tiene un propósito específico y se define al momento de crear la arquitectura. Para el desarrollo del sistema de gestión de órdenes del restaurante, se definió utilizar el patrón MVC (modelo-vista-controlador).

La selección del patrón MVC permite trabajar con aplicaciones que disponen de una estructura clara y organizada

El patrón MVC se basa en tres componentes esenciales que trabajan de manera conjunta, pero independiente, para lograr un diseño de *software* eficiente y modular. Cada uno de estos componentes desempeña un papel específico en el ciclo de vida de una aplicación, lo que simplifica la gestión de la lógica de negocio y la presentación de datos. El modelo se encarga de la lógica y el almacenamiento de datos, la vista se ocupa de la presentación visual y el controlador gestiona la interacción del usuario.

Para el prototipo, se va a aplicar el patrón MVC a todo el proyecto, haciendo una reestructuración sobre las clases existentes, separándolas en paquetes de Modelo donde están las clases que manipulan los datos (objetos, conexiones con la base de datos, etc), Vista que en este caso serian las páginas web en jsp para el cliente y Controlador que en este caso, serian las clases que interaccionan con las ordenes y las listas, añadiendo las clases nuevas relacionadas con la conexión con MySQL en los respectivos paquetes.

---

**Persistencia y consulta de datos en una base de datos MySQL. Esta actividad incluye establecer conexiones, realizar consultas, actualizar registros y presentar resultados en la interfaz.**

**Establecer conexiones**

En éste proyecto se creó la clase ConexionDB para realizar la conexión entre el programa en java y la base de datos MySQL, donde para la conexión si establecen parámetros como el nombre de la base de datos en donde realizaremos acciones en ella, como también la url en la que está, el usuario y contraseña con la que se accederá a la base de datos (en este caso es genérico, pero ya a la hora de entregar se generaría un usuario y contraseña más seguros), y se especifica el driver de jdbc que se utilizará, y el método conectar será quien con esos parámetros realice la conexión propiamente dicha.

```java
package Modelo;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ConexionDB {
    String bd= "bd_restaurante";
    String url = "jdbc:mysql://localhost:3306/";
    String user = "root";
    String pass = "";
    String driver = "com.mysql.cj.jdbc.Driver";
    Connection cx;

    public ConexionDB() {
    }
    public Connection conectar(){
        try {
            Class.forName(driver);
            cx = DriverManager.getConnection(url+bd, user, pass);
        } catch (SQLException |ClassNotFoundException ex) {
            System.out.println("FALLO LA CONEXION A DB " + bd);
            Logger.getLogger(ConexionDB.class.getName()).log(Level.SEVERE, null, ex);
        }
        return cx;
    }
```

---

**Realizar consultas**

Aquí se realiza la consulta a la base de datos sobre las bebidas existentes en ella, mediante el "*SELECT * FROM ítem WHERE ID_Categoria = 3*". Se lo hace mediante un PreparedStatement ya que la consulta lleva parámetros.

Luego lo que hace este método es obtener todos los campos/atributos de cada bebida traída del select, y crea los ítems con dichos campos/atributos para luego poder mostrarlos en el menú.

```
public List<Item> CargarBebidas(List<Item> Bebidas){
        String nombre_Item;
        String descripcion;
        double precio;
        int Id;
    try {
            try (PreparedStatement ps = cx.conectar().prepareStatement("SELECT * FROM item WHERE ID_Categoria = 3")) {
                ResultSet rs = ps.executeQuery();
                while(rs.next()){
                    Id = rs.getInt("ID_Item");
                    nombre_Item = rs.getString("Nombre");
                    descripcion = rs.getString("Descripcion");
                    precio = rs.getDouble("Precio");
                    Item it = new Item(Id,nombre_Item, descripcion, 1, precio);
                    Bebidas.add(it);
                }   }
        cx.desconectar();
    } catch (SQLException ex) {
        Logger.getLogger(FuncionesSQL.class.getName()).log(Level.SEVERE, null, ex);
    }


    return Bebidas;
}
```

---

**Actualizar registros**

Aquí se realiza la inserción de una orden realizada, a la base de datos, mediante el "*INSERT INTO orden(Codigo_mesa, Mozo_DNI, Fecha, Total) VALUES ( ? , ? , current_timestamp(), ? );*".

Los "?" en la sección de VALUES del INSERT, se rellenan con los setInt, y setDouble en las líneas de código siguientes con sus respectivos valores.

Se lo hace mediante un PreparedStatement ya que lleva parámetros, y se agrega *Statement.RETURN_GENERATED_KEYS* para almacenar la clave generada automáticamente en la inserción de ese registro, con el propósito de poder relacionar esta tabla con otra con la inserción de los ítems de la orden.

```
public void insert_orden(Orden orden, Mozo mozo) throws SQLException{
    int Cod_mesa = orden.getMesa();
    int DNI_Mozo = mozo.getDNI();
    Double Total = orden.Total;
    try {
        String sql_insert = "INSERT INTO orden(Codigo_mesa, Mozo_DNI, Fecha, Total) VALUES ( ? , ? , current_timestamp(), ? );";
        PreparedStatement PS = cx.conectar().prepareStatement(sql_insert, Statement.RETURN_GENERATED_KEYS);
        PS.setInt(1, Cod_mesa);
        PS.setInt(2, DNI_Mozo);
        PS.setDouble(3, Total);
        PS.executeUpdate();
        resultado = PS.getGeneratedKeys();
        if(resultado.next()){
            ultimoID = resultado.getInt(1);
        }
    } catch (SQLException ex) {
        Logger.getLogger(FuncionesSQL.class.getName()).log(Level.SEVERE, null, ex);
    }
    insert_items_orden(orden);
}
```

---

**Presentar resultados en la interfaz**

Aquí se muestra el prototipo de menú, con todos los ítems que están en la base de datos, mostrando el nombre, descripción y precio de cada uno.

## MENÚ

- ☐ Cafe (Molido) ---------- 1600.0
- ☐ Cafe (Cortado) ---------- 2600.0
- ☐ Té (Verde) ---------- 1600.0
- ☐ Té (Hiervas) ---------- 1600.0
- ☐ Coca-Cola (Botella de 1lts) ---------- 2000.0
- ☐ Coca-Cola (Botella de 500cc) ---------- 1500.0
- ☐ Sprite (Botella de 1lts) ---------- 2000.0
- ☐ Sprite (Botella de 500cc) ---------- 1500.0
- ☐ Cerveza (Botella de 1Lts) ---------- 4000.0
- ☐ Vino (Botella de 1Lts) ---------- 6000.0
- ☐ Empanada (De Carne) ---------- 1200.0
- ☐ Empanada (De jamon y queso) ---------- 1200.0
- ☐ Papas (Con cheddar) ---------- 5000.0
- ☐ Picada (Salamin, queso, mani jamon, bondiola) ---------- 8000.0
- ☐ Fideos (Fideos cinta) ---------- 3500.0
- ☐ Ñoquis (Con salsa roja) ---------- 3500.0
- ☐ Ñoquis (Con salsa 4 quesos) ---------- 4000.0
- ☐ Pizza (Comun) ---------- 6000.0
- ☐ Pizza (Especial) ---------- 8000.0
- ☐ Parrillada (Asado de tira, Morcilla, Chorizo, Corte de cerdo) ---------- 10000.0
- ☐ Entrecot (Bife de chorizo) ---------- 6500.0
- ☐ Milanesa (De Carne de vaca) ---------- 5000.0
- ☐ Suprema (De Carne de pollo) ---------- 4500.0
- ☐ Ensalada (Lechuga y tomate) ---------- 2500.0
- ☐ Ensalada (César) ---------- 2800.0
- ☐ Papas (Fritas) ---------- 3500.0
- ☐ Puré (De papa) ---------- 2600.0
- ☐ Helado (1 Bocha de Frutilla) ---------- 1500.0
- ☐ Helado (1 Bocha de Vainilla) ---------- 1500.0
- ☐ Ensalada De Fruta (Pera, uva, naranja, manzana) ---------- 1700.0
- ☐ Budin De Pan (Con crema) ---------- 2500.0
- ☐ Budin De Pan (Con dulce de leche) ---------- 2500.0

Enviar

---

**Correcta aplicación de excepciones para la interacción con la base de datos MySQL.**

Aquí la excepción se utiliza para capturar el cuando no se puede desconectar por algún error la conexión con la base de datos, mostrando un mensaje en consola y luego proporciona detalles sobre el error.

```
public void desconectar(){
    try {
        cx.close();
    } catch (SQLException ex) {
        System.out.println("FALLO LA DESCONEXION A DB " + bd);
        Logger.getLogger(ConexionDB.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

---

**Inclusión pertinente de clases abstractas o interfaces.**

Las clases abstractas son aquellas clases que dejan sin implementar algunos métodos o todos ellos, para que las clases derivadas de ella (Clases hijas) puedan proporcionar las implementaciones.

En la programación orientada a objetos, las clases abstractas se usan como clases bases de una jerarquía y representan la funcionalidad común de un conjunto diverso de tipos de objetos

En este proyecto considere no necesario la implementación de clases abstractas, ya que no tengo un grupo de objetos que hagan las mismas acciones de diferentes maneras.

---

**Utilización complementaria de arreglos y de la clase *ArrayList*.**

Aquí se hace uso de *ArrayList*, almacenando enteros que son los códigos de las mesas, que se obtienen de una consulta a la base de datos sobre los códigos de todas las mesas existentes en ella.

```
public ArrayList<Integer> CargarMesas(ArrayList<Integer> Mesas){

    try {
            try (PreparedStatement ps = cx.conectar().prepareStatement("SELECT * FROM mesa")) {
                ResultSet rs = ps.executeQuery();
                while(rs.next()){
                    int aux;
                    aux = rs.getInt("Codigo_mesa");
                    Mesas.add(aux);
                }   }
        cx.desconectar();
    } catch (SQLException ex) {
        Logger.getLogger(FuncionesSQL.class.getName()).log(Level.SEVERE, null, ex);
    }


    return Mesas;
}
```

Para luego iterar dicho *ArrayList* para verificar que el código insertado por el cliente que va a hacer la orden, está dentro de la base de datos, por ende es válido.

```
public boolean Verificar_Codigo(ArrayList<Integer> mesas){
  boolean verificacion_codigo = false;
  FuncionesSQL fsql = new FuncionesSQL();
  fsql.CargarMesas(mesas);
  for(Integer elemento : mesas){   //for each para iterar en el arreglo de mesas
      if(elemento == codigo){  //compara para ver si existe el codigo en el arreglo de mesas
          verificacion_codigo = true;
          break;
      }
  }
   return verificacion_codigo;}//devuelve el valor de verificacion_codigo
```

---

# REPOSITORIO

## LINK DE GITHUB DE LA BASE DE DATOS:

https://github.com/Giuliano-Kuhn/base_de_datos--APP_Restaurante.git

## Link de Github:

https://github.com/Giuliano-Kuhn/APP_restaurante.git

Los archivos .java están en la ruta:

**Modelo:** *src/main/java/Modelo*

**Vista:** *src/main/webapp*

**Controlador:** *src/main/java/Controlador*

Los archivos compilados (.class) están en la ruta:

**Modelo:** *target/classes/Modelo*

**Controlador:** *target/classes/Controlador*