

**FACULTAD DE INGENIERÍA ESCUELA PROFESIONAL DE
INGENIERÍA DE SISTEMAS**

**Implementación de un sistema
de pedidos automatizado y
cartilla virtual aplicada al
restaurante “Plaza mayor”**

AUTORES:

Chileno Santisteban, Jennifer Paola (orcid.org/0000-0001-9137-3352)

Leon Yaipen, Piero Daniel (orcid.org/0000-0003-1880-0084)

Esquives Zapata, Juan Jesus (orcid.org/0000-0002-6263-9745)

Casas Castillo, Marcia Verónica (orcid.org/0000-0002-7780-7844)

Asesor:

Mgstr. Guevara Jimenez, Jorge Alfredo (0000-0002-8459-9342)

Línea de Investigación:

Tecnología de la información y comunicación

LÍNEA DE RESPONSABILIDAD SOCIAL UNIVERSITARIA:

Desarrollo económico, empleo y emprendimiento

LIMA – PERÚ

2023

ÍNDICE

I. Introducción	1
1.1. Realidad problemática	2
1.2 Marco Teórico	3
1.2.1. Requerimiento funcional:	3
1.2.2. Requerimiento No Funcional	3
1.2.3. Atributos de calidad	3
1.2.4. Sistemas de información	4
1.2.5. Casos de uso	4
1.2.6. Diagrama de caso de uso	4
1.2.7. Diagramas de secuencia	4
1.4 Justificación	5
1.5. Diseño de investigación	6
1.6 Objetivos	7
1.7. Recolección de datos	7
II. Descripción y análisis	9
2.1 Requerimientos no funcionales	9
2.2 Requerimientos funcionales	10
2.3 Modelo de casos de uso del sistema	12
2.4 Especificaciones de caso de uso	13
2.5 Modelo de análisis RUP	16
2.6 Diagramas de secuencia	21
2.7 Prototipo	23
III. Conclusiones	25
IV. Anexos	26
V. Referencias bibliográficas	27

I. Introducción

En la era digital actual, la eficiencia y comodidad son elementos clave en el mundo de la gastronomía. La industria restaurantera ha evolucionado drásticamente con la adopción de tecnologías que mejoran la experiencia del cliente y optimizan la operación interna de los establecimientos. Uno de estos avances fundamentales es la implementación de sistemas de pedidos automatizados, los cuales revolucionan la forma en que los clientes realizan sus pedidos y cómo los restaurantes gestionan sus servicios. Estos sistemas no solo agilizan el proceso de toma de pedidos, sino que también permiten una mayor personalización y adaptabilidad a las preferencias individuales de los comensales.

La creación de un sistema de pedidos para un restaurante no solo se trata de implementar una tecnología avanzada, sino de diseñar una experiencia fluida y placentera para los clientes. Desde la comodidad de sus dispositivos móviles, los comensales pueden explorar el menú, personalizar sus órdenes y realizar pagos de manera sencilla. Además, los restaurantes pueden aprovechar estos sistemas para optimizar la gestión de inventario, analizar datos de consumo y anticipar las demandas de los clientes. Todo esto contribuye a una operación más eficiente y a la satisfacción constante de los clientes, elementos cruciales en la competitiva industria de la restauración.

El diseño y desarrollo de un sistema de pedidos efectivo implica un entendimiento profundo de las necesidades específicas del restaurante y de su clientela. Es esencial considerar aspectos como la capacidad de adaptación a diferentes tipos de cocina, la integración con sistemas de pago seguros y la interfaz fácil de usar para brindar una experiencia sin contratiempos. Asimismo, es crucial garantizar que el sistema cumpla con los estándares de seguridad y privacidad para proteger los datos sensibles de los clientes y mantener la confianza en la plataforma.

En este contexto, explicaremos en detalle la importancia de crear un sistema de pedidos eficiente para un restaurante, los beneficios tanto para los clientes como para los propietarios de establecimientos, y los aspectos clave a considerar en el proceso de diseño e implementación de este sistema. Este análisis nos permitirá comprender cómo la tecnología puede transformar la dinámica de la industria restaurantera, ofreciendo una experiencia de consumo mejorada y más ágil para todos los involucrados.

1.1. Realidad problemática

En los últimos años, se ha notado un notable impacto de los sistemas de información en Perú. Han revelado importantes oportunidades de crecimiento en diversos sectores económicos del país, incluyendo gastronomía, educación, salud y transporte, entre otros.

Este hecho subraya la importancia fundamental de los sistemas de información en respaldar el desarrollo de organizaciones que ofrecen productos o servicios. Su contribución radica en optimizar el procesamiento, análisis y organización de datos, facilitando la toma de decisiones que a su vez busca la satisfacción del cliente y, en última instancia, el crecimiento de la entidad.

Particularmente en la región Junín, especialmente en Huancayo, se está experimentando un notable avance en el ámbito de los sistemas de información. Este progreso se concentra especialmente en el sector gastronómico, esencial para el desarrollo regional. La destacada presencia de platos típicos en esta área atrae a los visitantes, pero persiste cierto descontento por parte de los consumidores.

Por lo tanto, el presente trabajo de investigación se enfocará en mejorar el sistema de atención al cliente de un establecimiento. El objetivo principal es optimizar la eficiencia en la gestión de pedidos, aprovechando las ventajas de la tecnología actual. Para ello, se propondrá la implementación de una cartilla virtual que permita a los clientes acceder de manera ágil y sencilla al catálogo de productos o servicios disponibles. Esta herramienta digital no sólo agilizará el proceso de solicitud, sino que también facilitará la navegación y elección de opciones para los consumidores. Además, se busca integrar funcionalidades que mejoren la comunicación y

retroalimentación entre el cliente y el establecimiento, contribuyendo así a una experiencia más satisfactoria y eficiente. Este enfoque se alinea con la demanda actual de soluciones tecnológicas que brinden comodidad y rapidez en las transacciones comerciales (Tovar R. , 2019, p.15).

1.2 Marco Teórico

1.2.1. Requerimiento funcional:

Por otro lado, como mencionó Montes y Brito (2012), los requerimientos funcionales consisten en las explicaciones de los servicios que el sistema ofrecerá y cómo reacciona ante entradas particulares. Estos pueden variar según el tipo de software y los posibles usuarios, y se enfocan en describir las capacidades del sistema, incluyendo cómo maneja las entradas, las salidas y las situaciones excepcionales. En ocasiones, también establecen limitaciones al indicar lo que el sistema no debe realizar (p. 44).

1.2.2. Requerimiento No Funcional

Los requerimientos no funcionales se concentran en la manera en que el software funciona y se relacionan con el comportamiento que se espera de la solución. Estos requisitos se ocupan de aspectos como el tiempo de funcionamiento en entornos específicos y las características operativas que el sistema debe tener, como capacidad, experiencia del usuario, disponibilidad, velocidad, seguridad y organización de la información (Wong, 2017, p. 52).

1.2.3. Sistemas de información

Son conjuntos organizados de elementos que permiten gestionar, procesar, almacenar y comunicar información de forma eficiente y segura. Los sistemas de información se componen de personas, procedimientos y tecnologías, que interactúan entre sí para cumplir con los objetivos de una organización. Los sistemas de información se utilizan para apoyar la operación, la gestión y la estrategia de una organización. (Davis y Olson, 1985).

1.2.4. Casos de uso

Un caso de uso es un elemento que detalla una secuencia de acciones que llevan a un resultado que es valioso y claramente observable. Estos casos proporcionan una forma de expresar los requisitos funcionales en el contexto de los procesos empresariales y del sistema. Pueden ser representados visualmente en un diagrama o mediante una descripción escrita en un documento (IBM, 2015, párr. 1).

1.2.5. Diagrama de caso de uso

El diagrama de casos de uso ilustra la interacción entre un usuario (representado como actor) y el sistema en desarrollo, presentando la estructura, naturaleza y secuencia de las operaciones o casos de uso. Un actor se describe como un rol que un individuo asume en relación con el sistema, lo que significa que no siempre se refiere a una persona específica, sino más bien a la función que desempeña en relación con el sistema (Manzanares, 2018, párr. 1).

1.2.6. Diagramas de secuencia

Los diagramas de secuencia son una categoría de diagramas de comportamiento utilizados para mostrar la secuencia de acciones en la ejecución de un proceso o caso de uso particular. Estos diagramas representan quién está llevando a cabo el proceso, qué objetos están implicados, cómo se comunican a través de mensajes y la duración de su interacción. Son una herramienta valiosa para definir los tiempos de respuesta ideales del software y para identificar y corregir errores al proporcionar una visualización del funcionamiento del sistema (Alava, 2015, párr. 1).

1.4 Justificación

Respecto a la justificación tenemos, la justificación práctica la cual presenta el caso de estudio con el fin de mejorar la eficiencia y la calidad del servicio en la industria de la restauración a través de la implementación de una cartilla virtual, lo que puede aumentar la satisfacción del cliente y la rentabilidad de los restaurantes, además de ofrecer soluciones para

desafíos operativos comunes. En el caso del restaurante "Plaza Mayor", se está observando cómo dicho establecimiento y otros de su tipo no cuentan con un sistema de registro de pedidos eficiente. Esto ha llevado a una atención al cliente deficiente debido a los registros manuales, lo que resulta en la pérdida de tiempo y clientes debido a las demoras en su servicio. Esta situación está afectando negativamente la capacidad de "Plaza Mayor" para proporcionar un servicio de calidad. Por esta razón, se presenta la propuesta de un software altamente eficiente, diseñado específicamente para "Plaza Mayor", con el objetivo de agilizar el proceso de registro de pedidos y facilitar la gestión de mesas dentro del restaurante. Además, con respecto a la justificación valorativa en este caso se centra en el reconocimiento de la importancia de mejorar la calidad de los servicios en la industria de restaurantes. La falta de un sistema eficiente de registro de pedidos ha llevado a pérdidas de tiempo y clientes, lo que impacta negativamente en la experiencia del cliente y en la competitividad de los restaurantes. Valoramos la satisfacción del cliente y la eficiencia operativa como aspectos cruciales para el éxito de estos negocios. La propuesta de un software eficiente busca elevar estos estándares y brindar un servicio de mayor calidad, lo cual consideramos fundamental para el bienestar de los clientes y el crecimiento de la industria de restaurantes en general. Finalmente, justificación académica para el caso de estudio del restaurante "Plaza Mayor" radica en su potencial para enriquecer el conocimiento académico en el campo de la gestión de restaurantes y la tecnología de la información aplicada a la industria de la hospitalidad. La falta de un sistema de registro de pedidos eficiente y sus consecuencias en la atención al cliente son problemas que requieren un análisis detallado y una solución innovadora. Este caso de estudio puede proporcionar información valiosa sobre cómo la implementación de un software personalizado puede abordar los desafíos operativos específicos de "Plaza Mayor" y mejorar la satisfacción del cliente en este contexto particular. Además, este estudio podría servir como un ejemplo académico de buenas prácticas para otros restaurantes y negocios similares que enfrentan desafíos similares en la gestión de operaciones y la atención al cliente.

1.5. Diseño de investigación

Dentro de la metodología de este diseño de investigación explicativo, se busca comprender en profundidad cómo la implementación del sistema de registro de pedidos y gestión de mesas impactará en la eficiencia operativa y la experiencia del cliente en el restaurante "Plaza Mayor". Se llevará a cabo un proceso detallado de diseño y desarrollo del software, asegurando que se adapte de manera óptima a las necesidades específicas de "Plaza Mayor". Además, se evaluará la efectividad y usabilidad del sistema a través de pruebas rigurosas y se recopilará retroalimentación del personal y los clientes del restaurante para garantizar que cumpla con sus expectativas y se alinee con los objetivos de mejora operativa. Este enfoque explicativo permitirá una comprensión más profunda de cómo la solución propuesta resuelve los desafíos identificados y contribuye a una gestión más eficiente y a una mejor experiencia del cliente en "Plaza Mayor".

1.6 Objetivos

a. Objetivo General

- Desarrollar e implementar un sistema de pedidos automatizado para un restaurante, considerando sus beneficios, desafíos y aspectos clave.

b. Objetivo Específico

- Desarrollar e implementar la funcionalidad para que el sistema pueda enviar de manera eficiente los pedidos a la cocina y permita marcar las mesas disponibles.
- Implementar un sistema de inicio de sesión que permita a los mozos acceder de manera segura. Además, desarrollar una funcionalidad que permita ver el estado actual del proceso de preparación de un plato en tiempo real.
- Incorporar una funcionalidad de chat en tiempo real entre los mozos y la cocina para agilizar la comunicación interna. Asimismo, integrar un sistema de generación automática de facturas o boletas para facilitar la gestión de transacciones y pagos.

1.7. Recolección de datos

La entrevista, como técnica investigativa, se utiliza para recopilar información de manera directa y personalizada a través de la comunicación oral. Durante este proceso, se exploran tanto los acontecimientos experimentados por la persona como sus percepciones subjetivas, incluyendo aspectos como creencias, actitudes, opiniones y valores, que guardan relación con el tema o situación objeto de estudio. Esta interacción cara a cara entre el investigador y el entrevistado permite profundizar en la comprensión del tema en cuestión y obtener insights valiosos para el análisis y desarrollo de la investigación (Hernandez, Garrido, Martin, Gomez , 2015, p.6). A continuación, se presentarán los puntos más importantes para poder tener una buena entrevista

- **Preparación y Planificación**

Antes de la entrevista, investiga y conoce a fondo el tema y la

persona que será entrevistada. Diseña un guión o lista de preguntas que te ayuden a obtener la información deseada. Define claramente los objetivos de la entrevista y la estructura que seguirás.

- Establecer un Ambiente Apropiado

Crea un entorno cómodo y acogedor para la entrevista. Asegúrate de que la persona entrevistada se sienta tranquila y en confianza para expresarse libremente. Evita distracciones y garantiza que el espacio permita una comunicación efectiva.

La conversación se centró en escuchar las preocupaciones y objetivos del restaurante "Plaza Mayor", sin limitarse exclusivamente a los requisitos funcionales. Se buscó obtener una comprensión completa de la situación actual y las áreas de mejora identificadas por la dirección del restaurante. Además de la entrevista, se llevaron a cabo observaciones en el lugar para contextualizar la información y verificar los datos recopilados. Esta combinación de técnicas de recopilación de datos proporcionó una visión integral de las necesidades y desafíos de "Plaza Mayor", lo que servirá como base sólida para el desarrollo del proyecto en su conjunto.

II. DESCRIPCIÓN Y ANÁLISIS DEL FENÓMENO ESTUDIADO

2.1. Descripción del caso de estudio

El caso de estudio se enfoca en el restaurante "Plaza Mayor", que está buscando mejorar tanto sus procesos internos como la satisfacción de sus clientes a través de la introducción de un sistema de registro de pedidos y gestión de mesas basado en software. "Plaza Mayor" ha experimentado desafíos relacionados con la eficiencia en la toma de pedidos, la administración de las mesas y la satisfacción de sus clientes debido a la ausencia de una solución tecnológica efectiva. Para abordar estos problemas, se ha realizado una investigación exhaustiva que incluye entrevistas con el personal de "Plaza Mayor" y observaciones directas en el lugar con el fin de comprender en detalle las necesidades y desafíos específicos que enfrenta el restaurante. Como resultado de esta investigación, se ha diseñado y desarrollado un software personalizado con el propósito de solucionar estos problemas y mejorar la operación general de "Plaza Mayor". El informe presenta los resultados anticipados de esta implementación y cómo se espera que influya positivamente en la eficiencia de las operaciones y la experiencia de los clientes en el restaurante "Plaza Mayor".

2.1 Requerimientos no funcionales

Item	Nombre	Descripción
1	Seguridad	Este requerimiento se enfoca en garantizar la integridad, confidencialidad y disponibilidad de los datos y del sistema en general. Incluye medidas como autenticación de usuarios, control de acceso, cifrado de datos y auditoría de actividades. Se deben implementar prácticas de seguridad sólidas para proteger el software contra amenazas y ataques.

2	Base de datos (SQL Server)	Este requerimiento se refiere a la elección de SQL Server como sistema de gestión de bases de datos. Implica que el software debe estar diseñado para interactuar eficazmente con SQL Server para almacenar, recuperar y gestionar datos de manera segura y eficiente.
3	Disponibilidad del local	Este requerimiento se enfoca en asegurar que un lugar físico, como una instalación o un edificio, esté disponible y accesible según un horario específico o circunstancias programadas. Puede aplicarse en diferentes contextos, como tiendas, hospitales, eventos, etc., y requiere planificación de horarios, mantenimiento y planes de contingencia.
4	Nestjs tecnología front	Este requerimiento indica que Nest.js, que es un framework de desarrollo de backend basado en Node.js, se utilizará para implementar parte de la lógica del servidor. Implica la necesidad de integrar Nest.js con tecnologías front-end (por ejemplo, Angular, React) para construir una aplicación completa con capacidades tanto en el lado del servidor como en el cliente.
5	Docker	Este requerimiento se refiere a la compatibilidad con la tecnología de contenedores Docker. Implica que el software debe poder ser empaquetado en contenedores Docker para facilitar su implementación y gestión en diferentes entornos, como desarrollo, pruebas y producción.
6	Plataforma de Vista	Este requerimiento se relaciona con la elección de la plataforma o tecnología utilizada para crear la interfaz de usuario (UI) o vista del software. Puede incluir plataformas como HTML/CSS para aplicaciones web o frameworks de desarrollo de aplicaciones móviles para aplicaciones móviles. Implica la necesidad de diseñar y desarrollar la UI en la plataforma especificada.

Tabla 1. *requerimientos no funcionales*

2.2 Requerimientos funcionales

Item	Nombre	Descripción
1	Registrar Pedidos	Los mozos pueden registrar pedidos de los clientes, seleccionando los platillos deseados de un menú y asignando la mesa correspondiente. Deben especificar detalles como la cantidad y observaciones adicionales.
2	Consultar Pedidos por mesa	Tanto los mozos como la cocina pueden consultar los pedidos pendientes para conocer los detalles de los platillos solicitados, su estado y la mesa a la que pertenecen.

3	Actualizar Pedido	Los mozos pueden actualizar los pedidos en caso de cambios solicitados por el cliente, como agregar o eliminar platillos, cambiar la cantidad o realizar ajustes en las observaciones.
4	Culminar Pedido	Los mozos pueden marcar un pedido como completado una vez que los platillos se han preparado y entregado al cliente.
5	Eliminar pedido	Los mozos serán capaces de poder eliminar algún pedido que se haya tenido que cancelar
6	Registrar Platillo	El personal de cocina puede registrar nuevos platillos en el sistema, ingresando detalles como nombre, descripción, precio y disponibilidad.
7	Actualizar Platillo	La cocina puede actualizar la información de los platillos existentes, como cambiar su precio, descripción o disponibilidad.
8	Eliminar Platillo	La cocina puede eliminar platillos que ya no estén disponibles en el menú.
9	Consultar Platillo	Tanto los mozos como la cocina pueden consultar la información de los platillos disponibles en el menú.
10	Registrar Mesas	Los mozos pueden registrar las mesas disponibles en el restaurante, asignándoles un número o nombre y especificando su capacidad.
11	Consultar Mesas	Los mozos pueden consultar la disponibilidad de mesas y asignar mesas a los clientes que llegan al restaurante.
12	Actualizar mesas	Los mozos pueden actualizar la información de las mesas, como cambiar su estado (ocupado o desocupado) o la capacidad.
14	Eliminar mesas	Los mozos pueden eliminar mesas que ya no están en uso o que han sido reubicadas.
15	Registro de categoría	El sistema debe permitir registrar categoría
16	Actualizar categoría	El sistema debe permitir actualizar categoría
17	Consultar categoría	El sistema debe permitir consultar categoría
18	Desactivar categoría	El sistema debe permitir desactivar categoría
19	Registrar	El sistema debe permitir registrar empleado

	empleado	
20	Consultar empleado	El sistema debe permitir consultar un empleado
21	Actualizar empleado	El sistema debe permitir actualizar un empleado
22	Desactivar empleado	El sistema debe permitir desactivar un empleado
23	Notificación de pedido	La cocina recibe notificaciones en tiempo real cuando se registra un nuevo pedido o cuando se actualiza un pedido existente, lo que les permite comenzar a preparar los platillos de inmediato.
24	Generar QR (Yape, Plin, etc)	Los clientes pueden generar códigos QR para realizar pagos a través de aplicaciones de pago como Yape o Plin. Estos códigos se generan automáticamente y se vinculan con la factura del pedido.
25	Generar boletas o facturas	Los mozos pueden generar boletas o facturas para los pedidos completados y entregárselas a los clientes. Estos documentos contienen detalles de los platillos consumidos y el monto total a pagar.
26	Reporte de ventas diarias	El sistema genera automáticamente un informe de las ventas diarias, que incluye el total de ingresos, el número de pedidos, y otros datos relevantes para la gestión financiera
27	Reporte de productos más vendidos	El sistema genera un informe que muestra los platillos más vendidos durante un período de tiempo específico, lo que ayuda a la gerencia a tomar decisiones sobre el menú y las estrategias de promoción.
28	Implementación de búsqueda de información de clientes	El sistema permite hacer búsquedas por medio de DNI de los clientes
29	Carga de imágenes con eliminación de fondo	El sistema permite la carga de imágenes junto a la eliminación del fondo de esta misma para una mejora en la visualización de cada platillo.

Tabla 2. *requerimientos funcionales*

2.3 Actores

- Mozos
- Cocina
- Personal de cocina

- Clientes
- Sistema (para notificaciones, generación de informes, gestión de categorías, empleados y más)

2.3 Límites del sistema

El sistema se conecta con los actores por medio de una interfaz de usuario, que toma la forma de una aplicación de escritorio. Además, establece comunicación con una base de datos SQL para almacenar y recuperar datos relevantes, como la información de los clientes y los registros de los pedidos de alquiler.

2.3 Diagrama de casos de uso del sistema

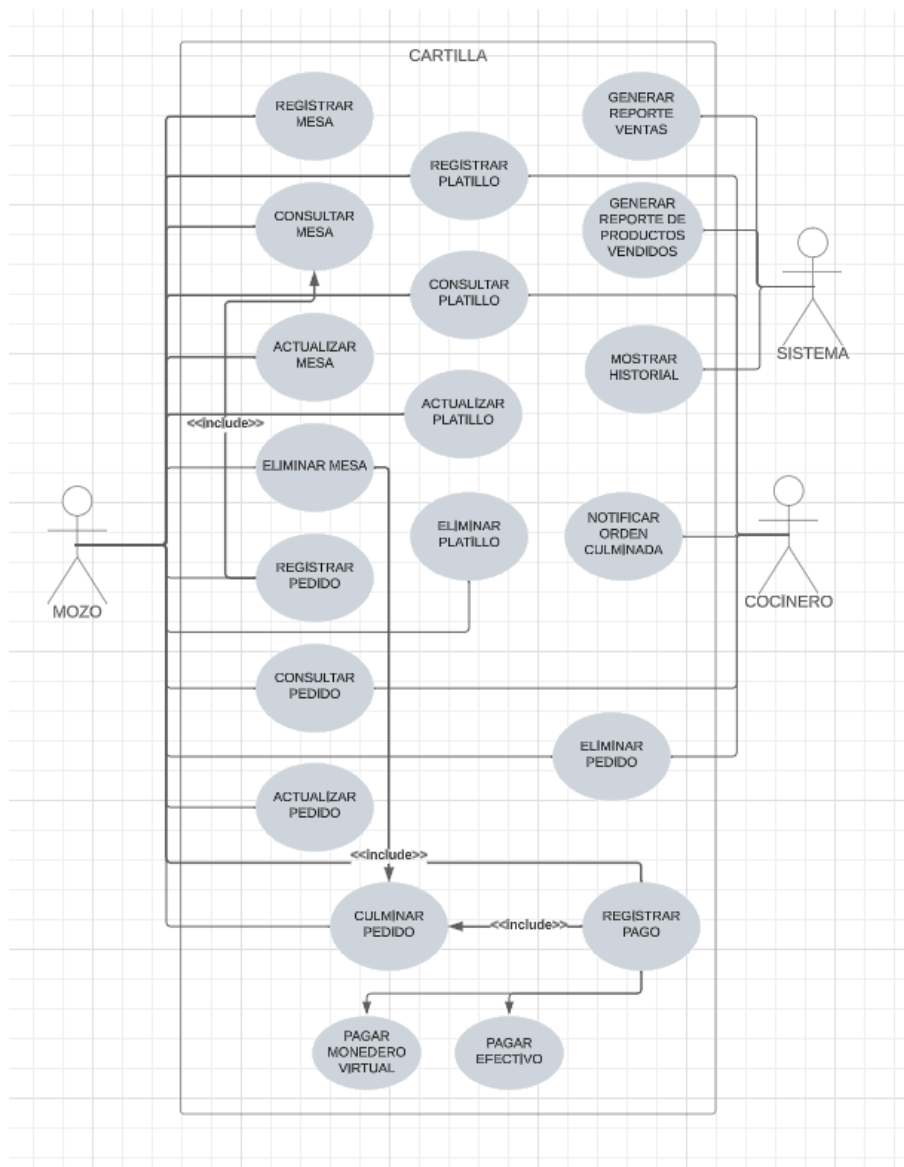


Figura 1. Diagrama de casos de uso del sistema

2.4 Especificaciones de caso de uso

IDCU	CU-GC-001		
NOMBRE CU	El sistema debe implementar registro de pedidos		
CREADO POR	Esquivel Leon Chileno Casas	Actualizado por	Grupo A
FECHA DE CREACIÓN	14/09/2023	Fecha de la última actualización	Próximamente
ACTORES	Mozo y Cocinero		
DESCRIPCIÓN	Este caso de uso describe el proceso de registro de pedidos por parte de los mozos en un restaurante. Los mozos toman los pedidos de los clientes, seleccionan los platillos deseados, especifican la cantidad y cualquier observación adicional, y asignan el pedido a una mesa		
DISPARADOR	Este caso de uso se dispara cuando un cliente realiza un pedido de comida en el restaurante.		
PRE-CONDICIONES	- El mozo debe estar autenticado en el sistema. - Deben existir mesas registradas en el sistema.		
POST-CONDICIONES	El pedido se registra en el sistema con todos los detalles necesarios. - El estado de la mesa se actualiza a "ocupada".		

FLUJO NORMAL	<ol style="list-style-type: none"> 1. El mozo inicia sesión en el sistema. 2. El mozo identifica la mesa a la que se le va a tomar el pedido. 3. El mozo selecciona los platillos del menú que el cliente ha pedido. 4. Para cada platillo seleccionado, el mozo indica la cantidad y puede agregar observaciones. 5. El mozo confirma el pedido y lo registra en el sistema
FLUJO ALTERNATIVO	<ul style="list-style-type: none"> - Si el mozo no está autenticado en el sistema: <ol style="list-style-type: none"> 1. El sistema muestra un mensaje de error. 2. El flujo vuelve al paso 1 (iniciar sesión). - Si no hay mesas registradas en el sistema: <ol style="list-style-type: none"> 1. El mozo no puede asignar un pedido a una mesa. 2. El flujo se detiene, y se debe registrar al menos una mesa antes de continuar.
EXCEPCIONES	<p>E1: Si la mesa seleccionada no está en estado "disponible", el sistema muestra un error y solicita al mozo que elija una mesa que esté disponible.</p> <p>E2: Si un plato seleccionado no está disponible en el menú, el sistema notifica al mozo y le pide que elija un plato diferente.</p> <p>E3: Si la cantidad de un plato se ingresa como cero o un número negativo, el sistema le pide al mozo que ingrese una cantidad válida.</p> <p>E4: Si hay una falla en el sistema o un error durante el proceso de registro del pedido, el sistema muestra un mensaje de error y permite al mozo volver a intentarlo o ponerse en contacto con el soporte.</p>
INCLUSIONES	<p>I1: El sistema incluye la capacidad para que el mozo vea el estado actual de cada mesa (disponible, ocupada, etc.).</p> <p>I2: El sistema incluye la capacidad para que el mozo vea el menú con el estado de disponibilidad de los platos.</p>

	I3: El sistema incluye una funcionalidad para calcular automáticamente el monto total del pedido en función de los platos seleccionados y las cantidades.
PRIORIDAD	Esencial
FRECUENCIA DE uso	Se prevé que este caso de uso tenga una alta frecuencia de uso, ocurriendo múltiples veces a lo largo de las horas operativas del restaurante a medida que los clientes realizan pedidos.
REGLAS De NEGOCIO	<ul style="list-style-type: none"> - El mozo solo puede registrar pedidos para mesas que estén en estado "disponible" en el sistema. - Los platillos deben estar disponibles en el menú para que el mozo los seleccione. - La cantidad de platillos debe ser un número positivo. - El sistema debe calcular automáticamente el total del pedido.
REQUISITOS ESPECIALES	<p>RE1: El sistema debe contar con una interfaz receptiva y fácil de usar, optimizada para tanto computadoras de escritorio como dispositivos móviles, para adaptarse a los diferentes dispositivos utilizados por los mozos.</p> <p>RE2: El sistema debe garantizar la seguridad y privacidad de los datos, especialmente en lo que respecta a la información de los clientes y los detalles de los pedidos.</p> <p>RE3: El sistema debe contar con mecanismos de copia de seguridad y recuperación para proteger contra la pérdida de datos o fallas del sistema, asegurando una interrupción mínima en las operaciones del restaurante.</p> <p>RE4: El sistema debe ser capaz de integrarse con otros módulos del sistema de gestión del restaurante, como facturación e inventario, para proporcionar una experiencia fluida para el personal y los clientes.</p>
ASUNCIONES	<p>Se asume que el sistema de gestión de restaurantes está disponible y funcionando correctamente.</p> <ul style="list-style-type: none"> - Se asume que los mozos están debidamente capacitados para utilizar el sistema y tomar pedidos. - Se asume que los clientes han ocupado una mesa en el restaurante antes de que se registre un pedido.

	<ul style="list-style-type: none"> - Se asume que el menú de platillos ha sido configurado previamente en el sistema y está actualizado.
NOTAS	<p>Este caso de uso puede ser parte de un sistema más amplio de gestión de restaurantes que incluye funciones adicionales como facturación, cocina, gestión de inventario, etc.</p> <ul style="list-style-type: none"> - La cantidad de mesas, platillos y mozos registrados en el sistema puede variar según las necesidades del restaurante. - El flujo normal no contempla el proceso de pago, que sería gestionado por otro caso de uso en el sistema. - La interfaz de usuario puede variar según la implementación específica del sistema.
PROTOTIPO	<ul style="list-style-type: none"> - El prototipo muestra una interfaz de usuario donde el mozo puede seleccionar platillos del menú y asignarlos a una mesa, especificando la cantidad y observaciones si es necesario. El prototipo debe reflejar la apariencia y la interacción prevista en el sistema.

Tabla 3. *Especificaciones de caso de uso*

2.5 Modelo de análisis de clases

- Registrar pedidos:

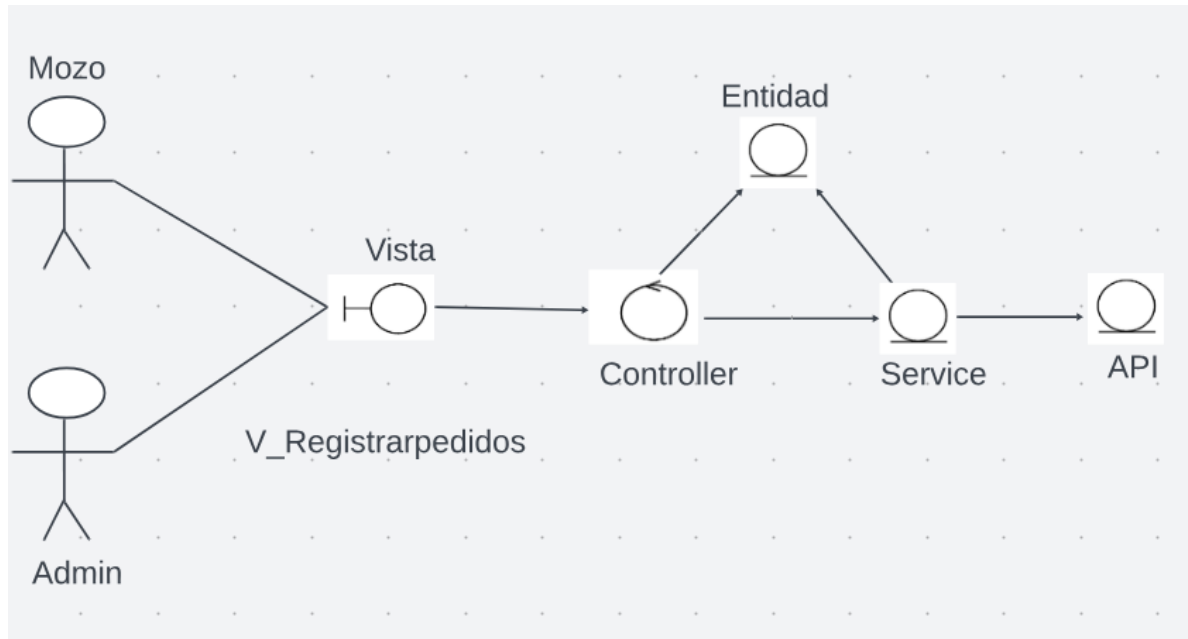


Figura 2 . Registro de pedidos

- Consultar pedidos:

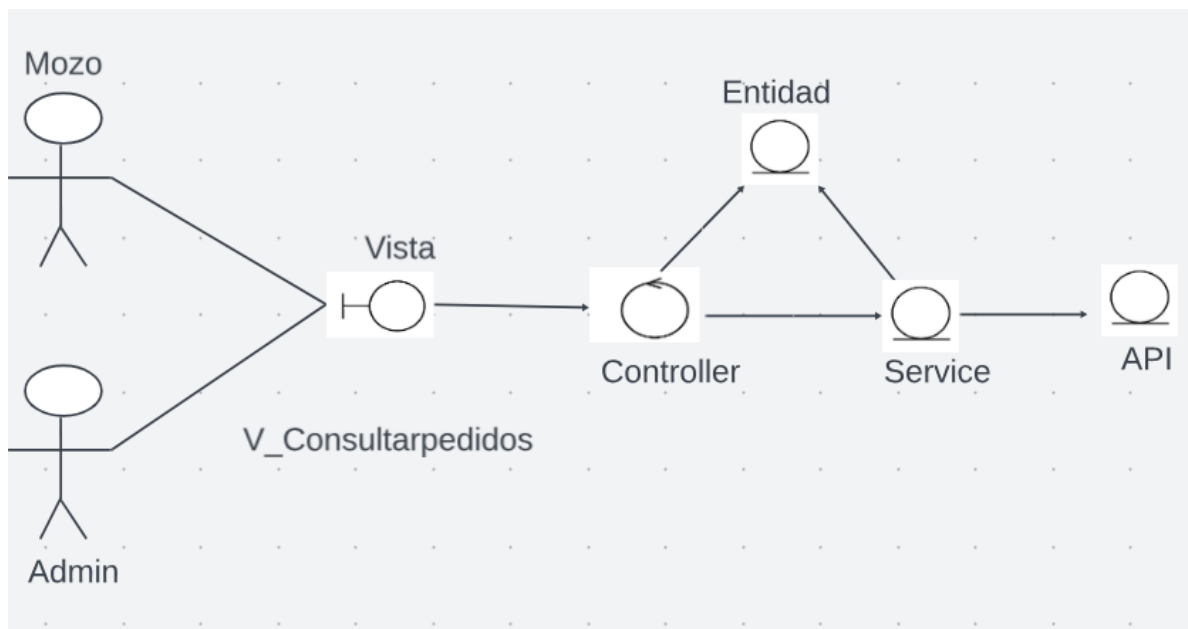


Figura 3 . Consultar pedidos

- **Eliminar pedidos:**

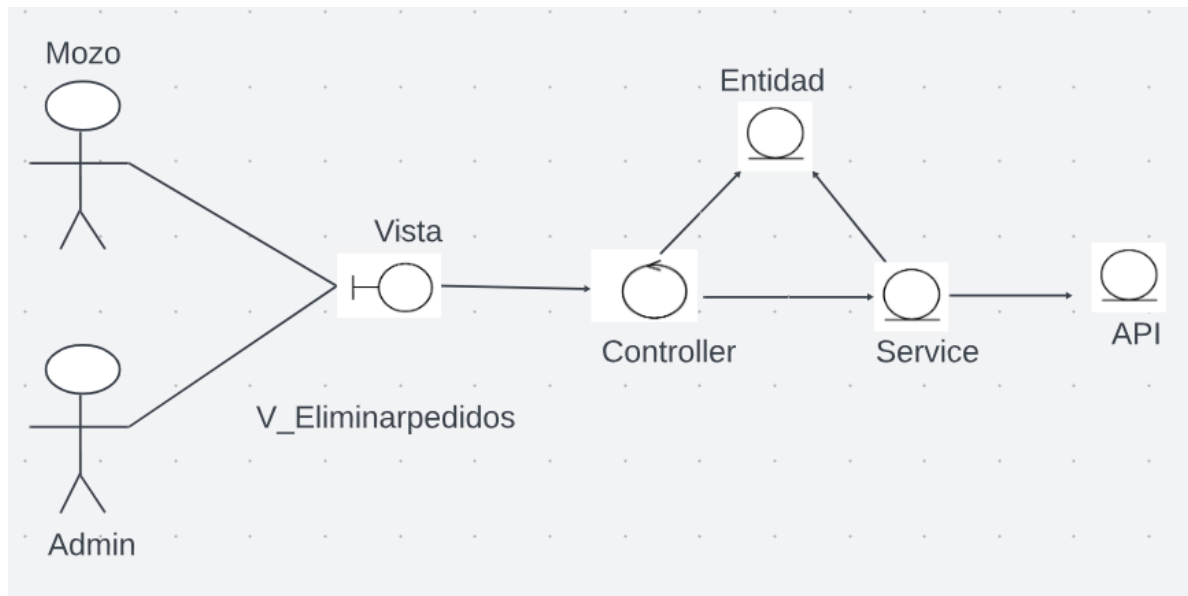


Figura 4 . Eliminar pedidos

- **Actualizar pedidos:**

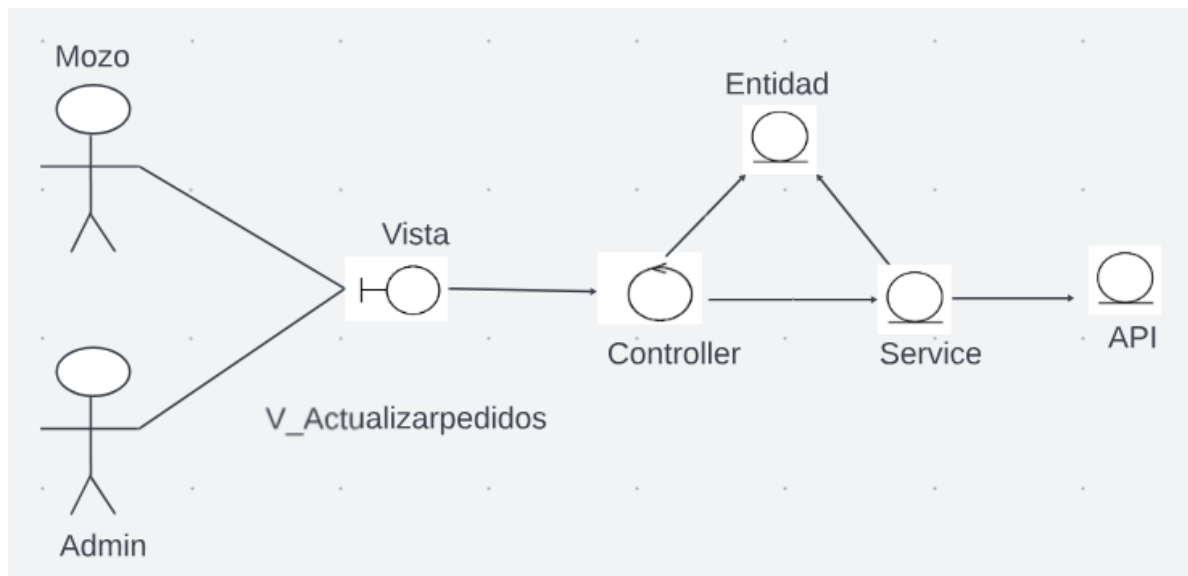


Figura 5 . Actualizar pedidos

- **Registrar y actualizar los platillos:**

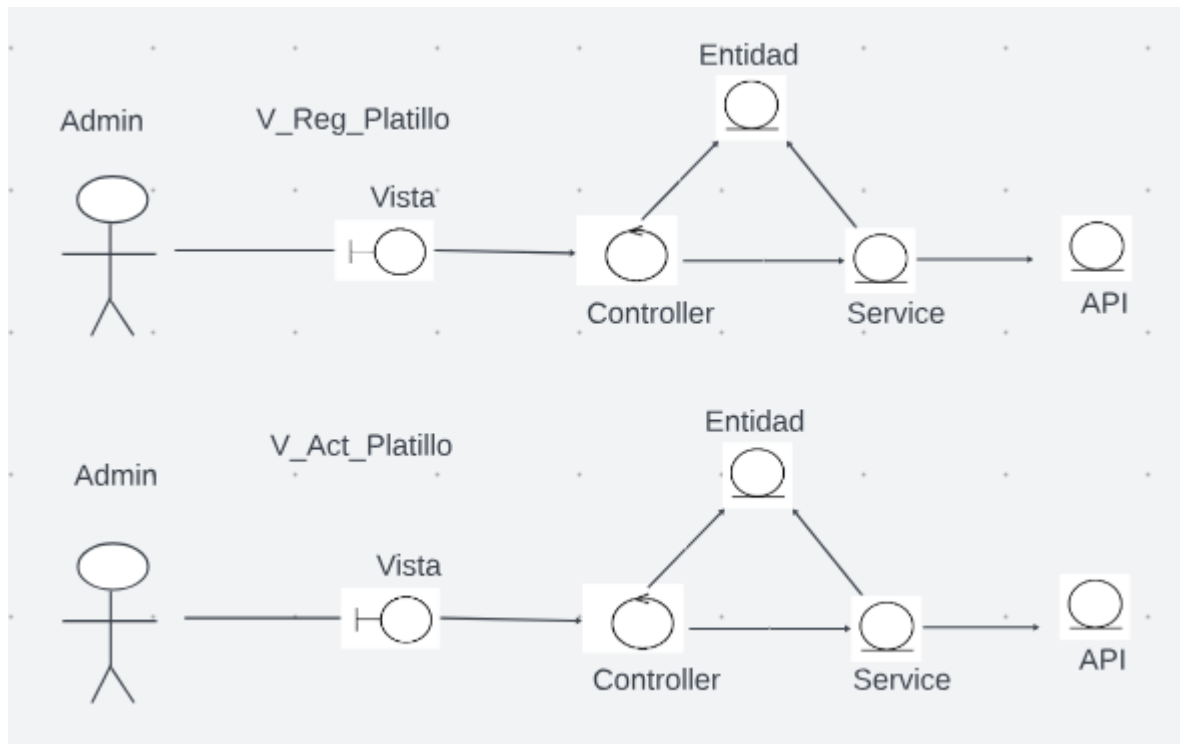


Figura 6 . Registrar y actualizar los platillos

- **Remove y consultar platillo:**

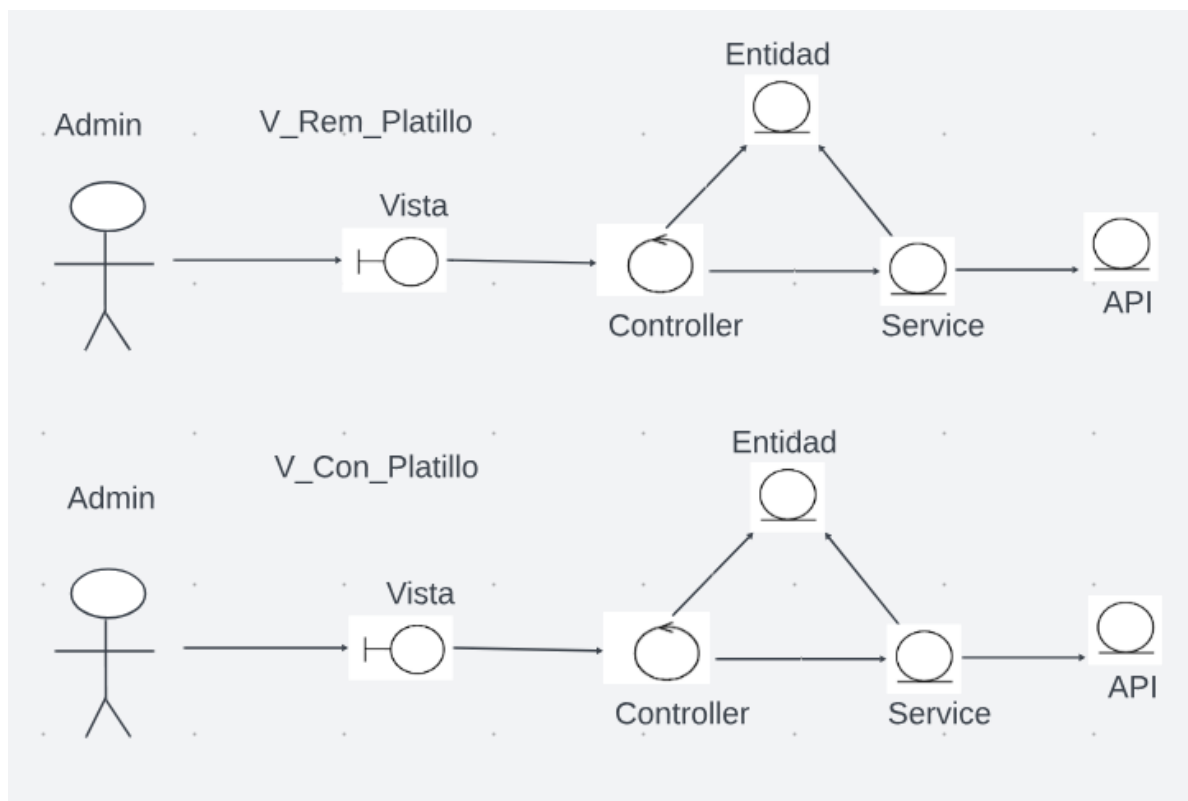


Figura 7 . Remove y consultar platillo

- **Registrar mesas:**

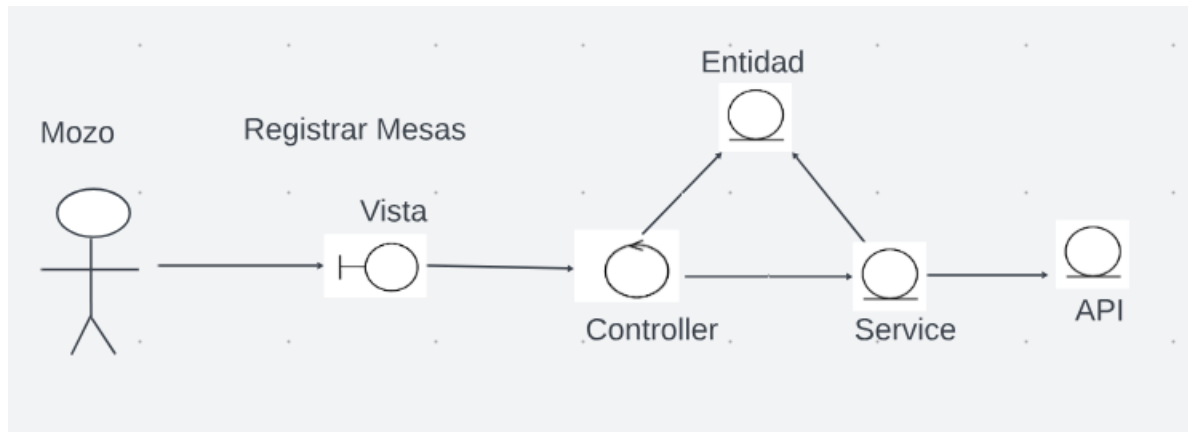


Figura 8 . Registrar mesas

- **Consultar mesas:**

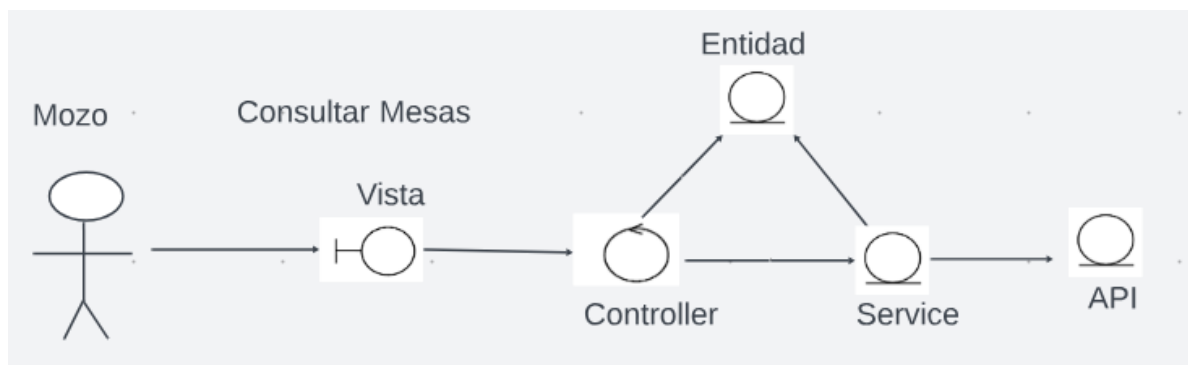


Figura 9 . Consultar mesas

- **Actualizar mesas:**

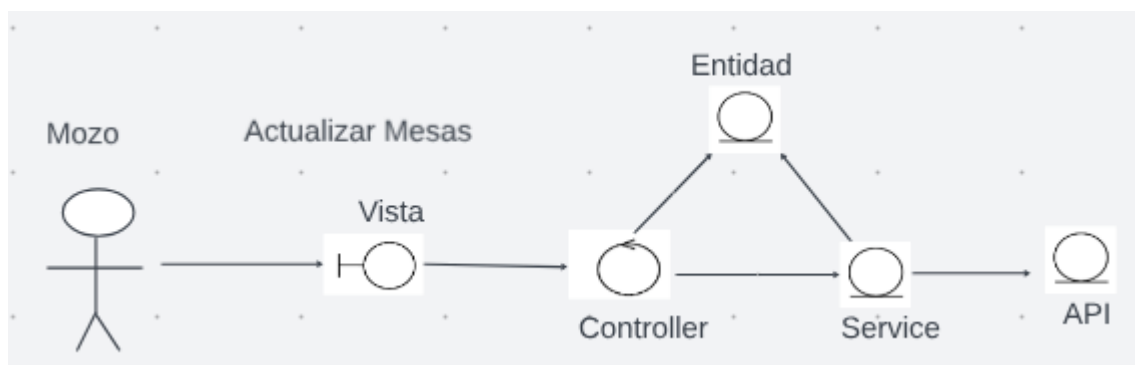


Figura 10. Actualizar mesas

- **Eliminar mesas:**

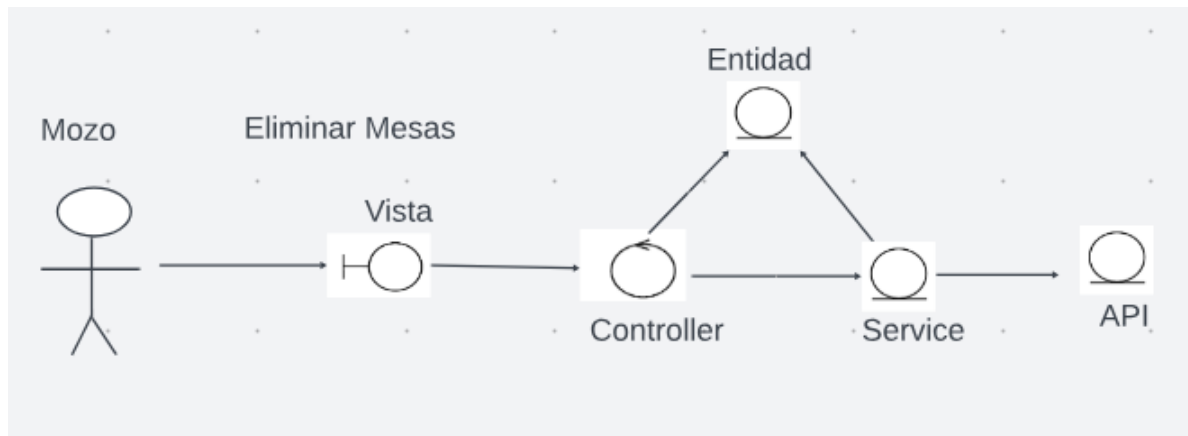


Figura 11. *Eliminar mesas*

2.6 Diagramas de secuencia

- **Pedidos:**

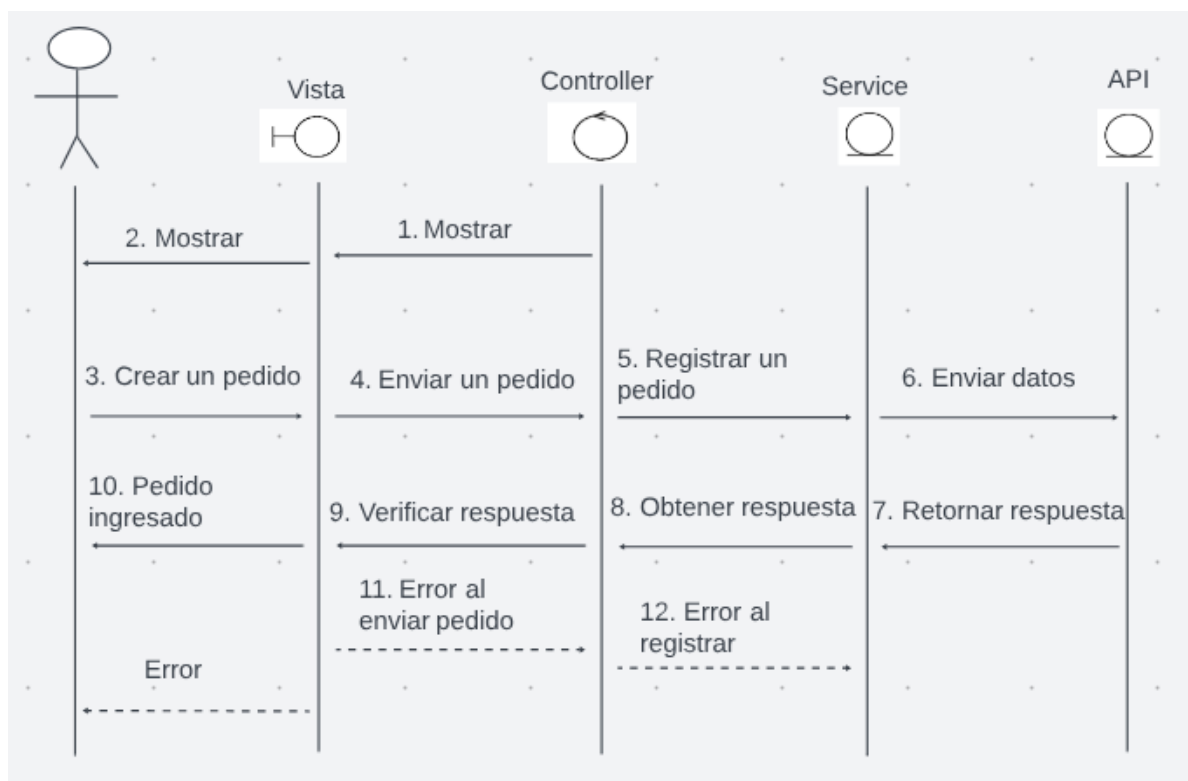


Figura 12. *Pedidos*

- **Platillos:**

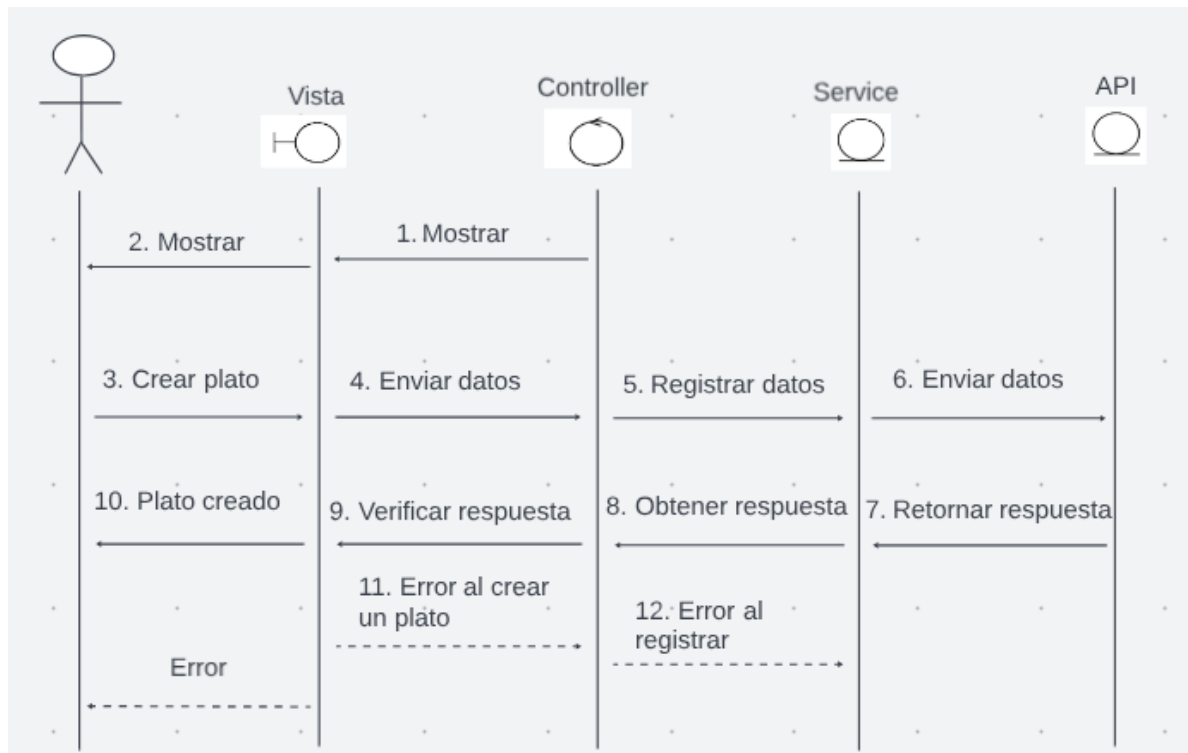


Figura 13. *Platillos*

- **Mesas:**

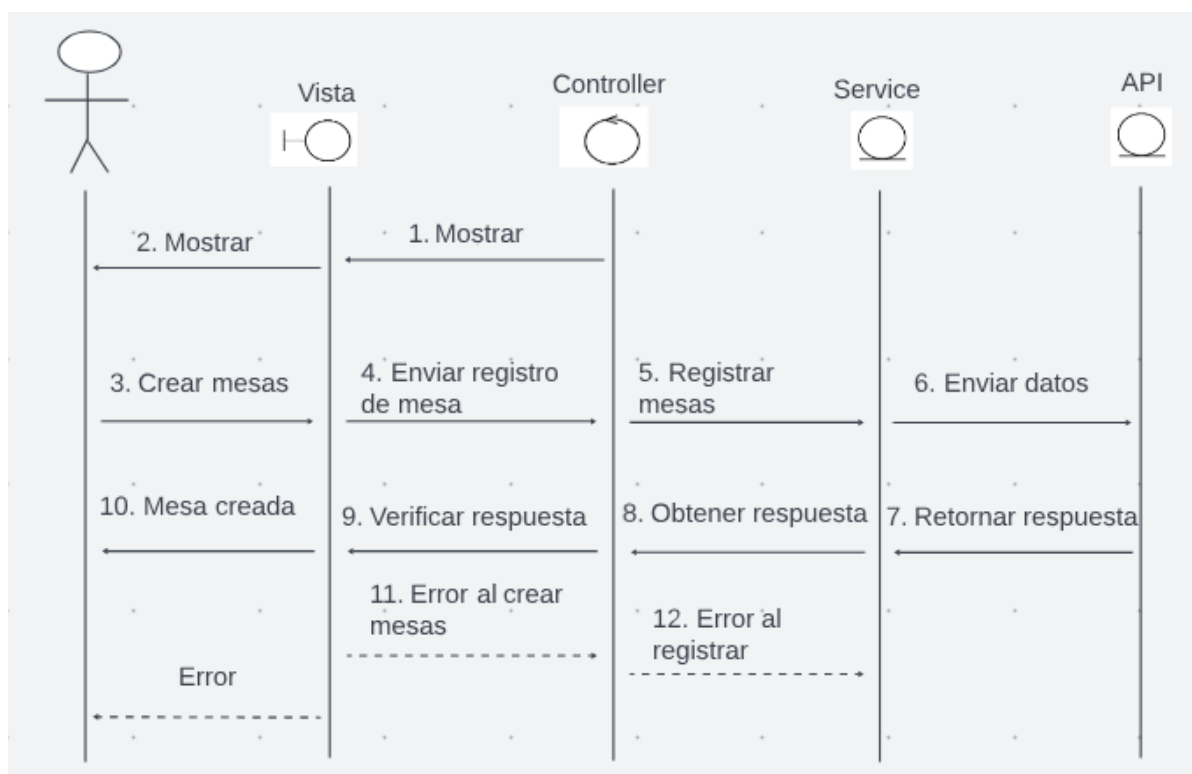


Figura 14. *Mesas*

2.7 Prototipo

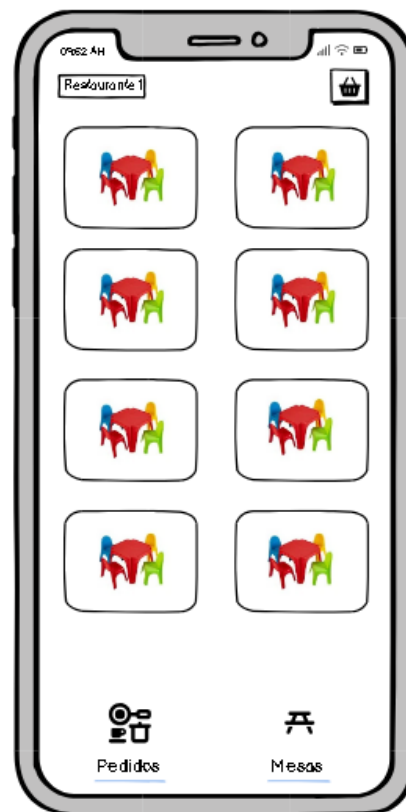


Figura 15. Prototipo 1

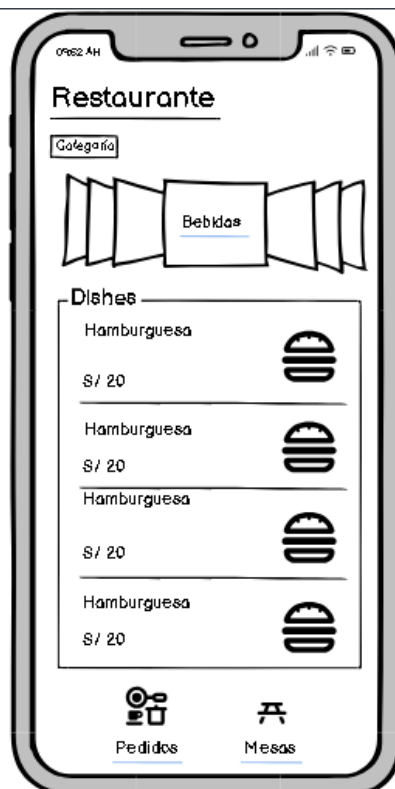


Figura 16. Prototipo 2

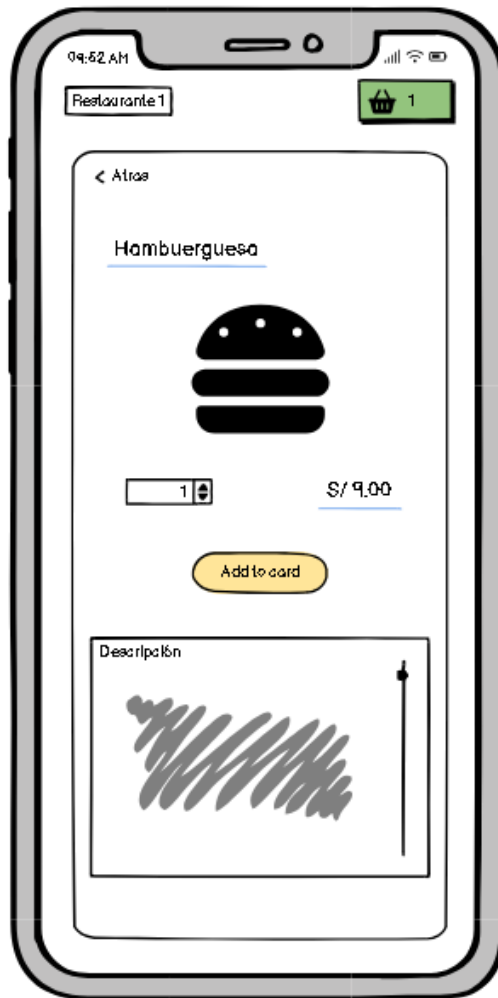


Figura 17. Prototipo 3

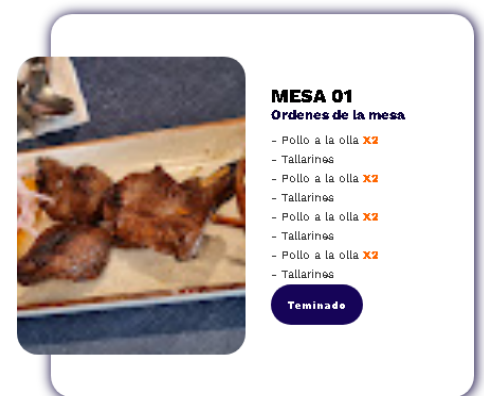


Figura 18. Prototipo 4

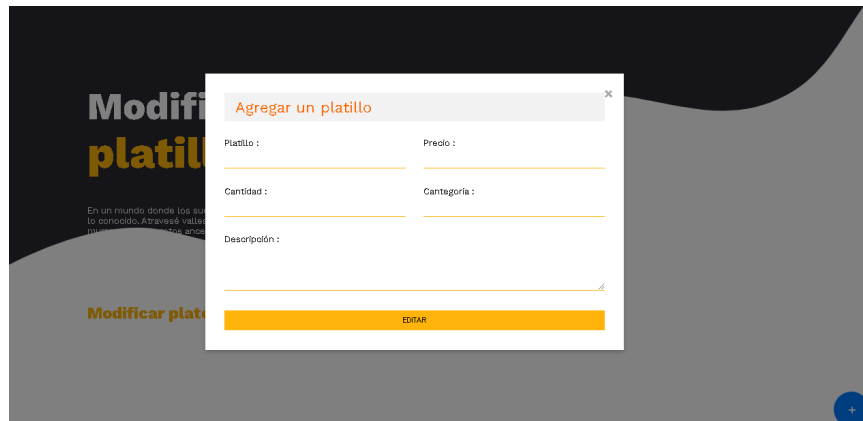


Figura 19. Prototipo 5

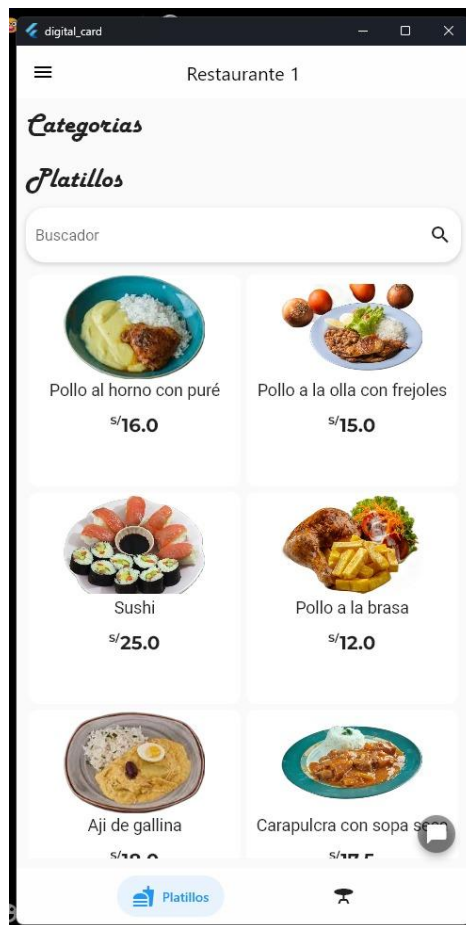


Figura 20. platillos

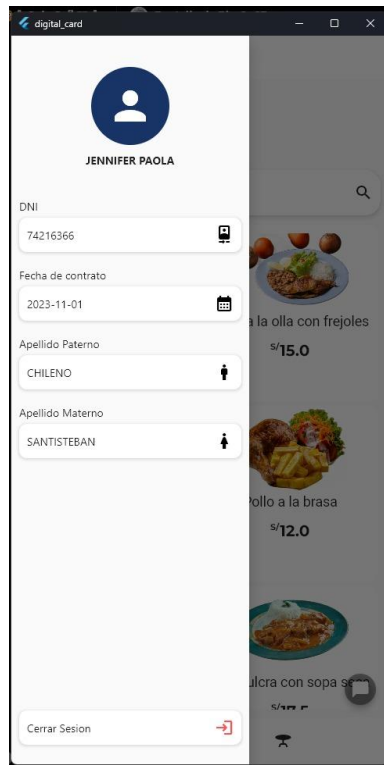


Figura 21. *Empleado*



Figura 22. *iniciar sesión*

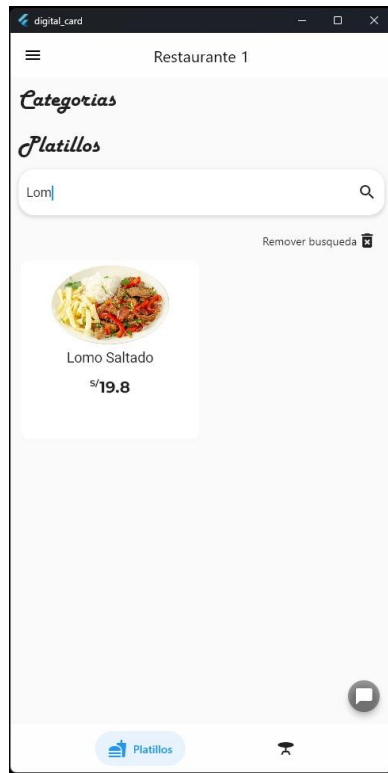


Figura 23. *platillo registrado*

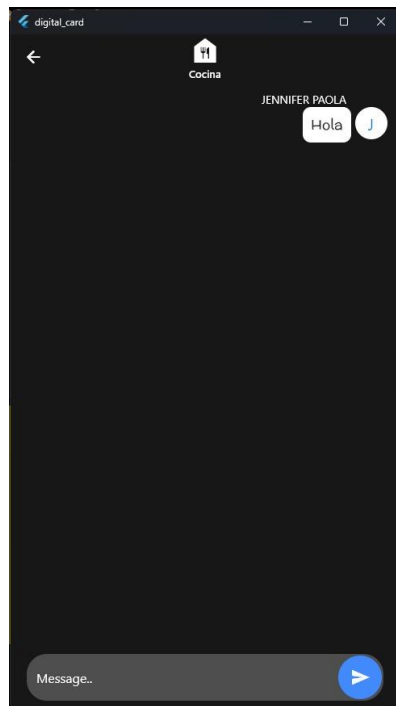


Figura 24. *Chat de empleados*



Figura 25. Mesas

Capítulo II:

Incremento 01:

CRUD DISHES

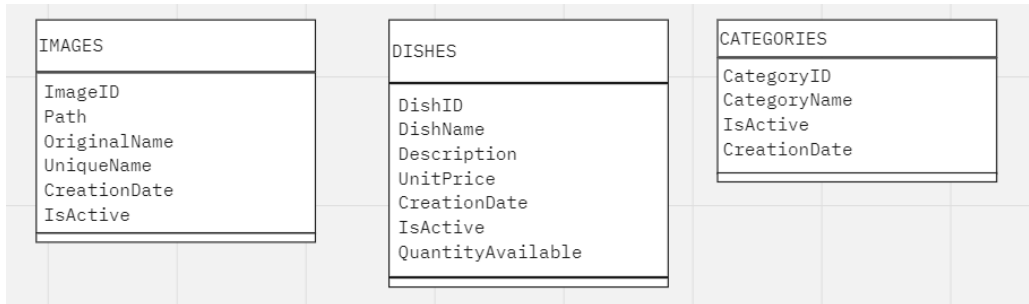


Figura 26. diagrama de entidades

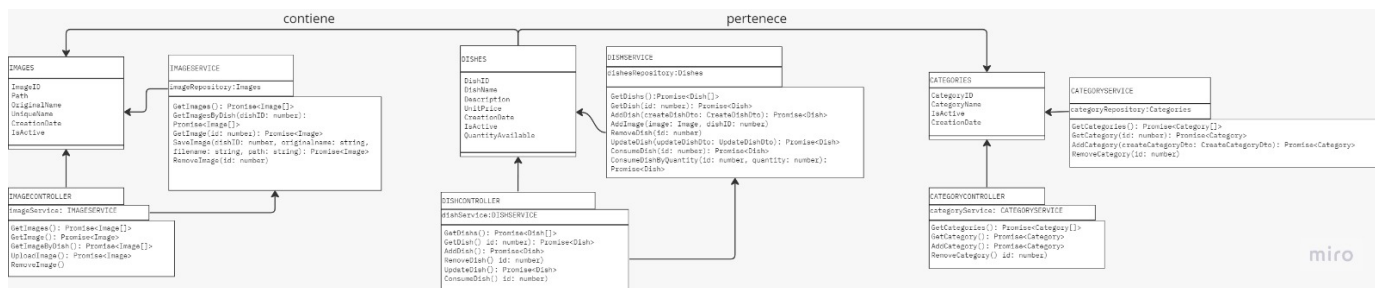


Figura 27. diagrama de entidades

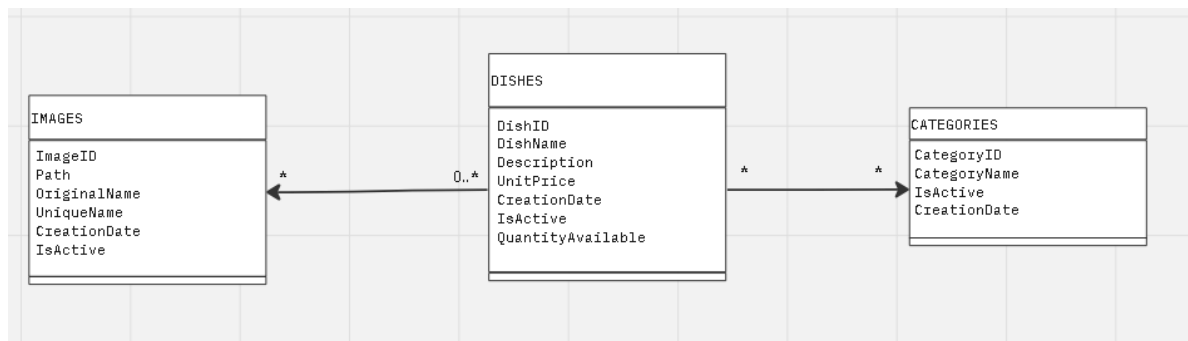


Figura 28. diagrama de entidades con cardinalidad

CLASE dish.entity:

```
7  ✓ export class Dish {  
8      @PrimaryGeneratedColumn({ name: "DishID" })  
9      DishID: number  
10  
11      @Column({  
12          type: "varchar",  
13          name: "DishName",  
14          length: 150,  
15          nullable: false  
16      })  
17      @Index({ unique: false })  
18      DishName: string  
19  
20      @Column({  
21          type: "text",  
22          name: "Description",  
23          nullable: true  
24      })  
25      Description?: string  
26  
27      @Column({  
28          name: "UnitPrice",  
29          type: "money",  
30          nullable: false  
31      })  
32      UnitPrice: number  
33  
34      @Column({  
35          name: "CreationDate",  
36          type: "date",  
37          nullable: true,  
38          default: () => "SYSDATETIME()",  
39      })  
40      CreationDate?: string  
41  
42      @Column({  
43          name: "IsActive",  
44          type: "bit",  
45          nullable: true,  
46          default: 1,  
47      })  
48      IsActive?: number  
49  
50  
51      @OneToMany(() => Image, (image) => image.Dish, { nullable: true, eager: true })  
52      Images?: Image[]  
53  
54      @Column({  
55          name: "QuantityAvailable",  
56          type: "int",  
57          nullable: true,  
58          default: 0  
59      })  
60      QuantityAvailable?: number  
61  
62  }
```

Figura 29. código dish.entity

Clase dish.controller:

```
import { Body, Controller, Delete, Get, Param, ParseIntPipe, Post, Put } from "@nestjs/common";
import { DishService } from "../services/dish.service";
import { Dish } from "../entities/dish.entity";
import { CreateDishDto } from "../dtos/create-dish.dto";
import { UpdateDishDto } from "../dtos/update-dish.dto";

@Controller("dishes")
✓ export class DishController {
  constructor(private readonly dishService: DishService) { }

  @Get()
  GetDishes(): Promise<Dish[]> {
    return this.dishService.GetDishes()
  }

  @Get("/:id")
  GetDish(@Param("id", ParseIntPipe) id: number): Promise<Dish> {
    return this.dishService.GetDish(id)
  }

  @Get("search/category/:category")
  GetDishesByCategory(@Param("category") category: string): Promise<Dish[]> {
    return this.dishService.GetDishesByCategory(category);
  }

  @Get("search/name/:name")
  GetDishesByName(@Param("name") name: string): Promise<Dish[]> {
    return this.dishService.GetDishesByName(name);
  }

  @Post()
  AddDish(@Body() createDishDto: CreateDishDto): Promise<Dish> {
    return this.dishService.AddDish(createDishDto)
  }

  @Delete("/:id")
  RemoveDish(@Param("id", ParseIntPipe) id: number) {
    this.dishService.RemoveDish(id)
  }

  @Put()
  UpdateDish(@Body() updateDishDto: UpdateDishDto): Promise<Dish> {
    return this.dishService.UpdateDish(updateDishDto);
  }

  @Put("/:id")
  ConsumeDish(@Param("id", ParseIntPipe) id: number) {
    this.dishService.ConsumeDish(id)
  }
}
```

Figura 30. código dish.controller

Clase dish.service:

```
@Injectable()
export class DishService {
  constructor(
    @InjectRepository(Dish)
    private dishRepository: Repository<Dish>,
    private readonly categoryService: CategoryService
  ) { }

  async GetDishes(): Promise<Dish[]> {
    let dishes = await this.dishRepository.find({
      where: { IsActive: 1 }, relations: { Categories: true }
    })
    return dishes
  }

  async GetDish(id: number): Promise<Dish> {
    let dish = await this.dishRepository.findOne({ where: { DishID: id, IsActive: 1 } })
    if (!dish) throw new NotFoundException('No se encontro el platillo con el id: '${id}''')
    return dish
  }

  async GetDishesByCategory(category: string): Promise<Dish[]> {
    let dishes = await this.dishRepository.find({ where: { Categories: { CategoryName: category }, IsActive: 1 } })
    if (dishes.length === 0) throw new NotFoundException('No se encontraron platillos con esa categoria')
    return dishes;
  }

  async GetDishesByName(name: string): Promise<Dish[]> {
    let dishes = await this.dishRepository.find({ where: { IsActive: 1, DishName: Like(`%${name}%`) } })
    if (dishes.length === 0) throw new NotFoundException('No se encontraron platillos con dicho nombre')
    return dishes;
  }

  async AddDish(createDishDto: CreateDishDto): Promise<Dish> {
    let categories = []
    if (createDishDto.CategoriesID) {
      categories = createDishDto.CategoriesID.map((categoryID) => this.categoryService.GetCategory(categoryID));
      console.log(categories);
    }
    let newDish = this.dishRepository.create(createDishDto)
    newDish.Categories = await Promise.all(categories)
    return await this.dishRepository.save(newDish)
  }

  async AddImage(image: Image, dishID: number) {
    let dish = await this.GetDish(dishID);
    if (!dish.Images) dish.Images = []
    dish.Images.push(image);
    await this.dishRepository.save(dish);
  }
}
```

```

async RemoveDish(id: number) {
    let dish = await this.GetDish(id)
    dish.IsActive = 0
    await this.dishRepository.save(dish)
}

async UpdateDish(updateDishDto: UpdateDishDto): Promise<Dish> {
    let dish = await this.GetDish(updateDishDto.DishID)
    Object.assign(dish, updateDishDto)
    return await this.dishRepository.save(dish)
}

async ConsumeDish(id: number): Promise<Dish> {
    let dish = await this.GetDish(id)
    if (dish.QuantityAvailable != null && dish.QuantityAvailable > 0) {
        --dish.QuantityAvailable
        return await this.dishRepository.save(dish)
    }
    return null
}

async ConsumeDishByQuantity(id: number, quantity: number): Promise<Dish> {
    let dish = await this.GetDish(id)
    if (dish.QuantityAvailable != null && dish.QuantityAvailable > 0 && quantity <= dish.QuantityAvailable) {
        dish.QuantityAvailable -= quantity
        return await this.dishRepository.save(dish)
    }
    return null
}

```

Figura 31. *código dish.service*

Clase image.entity:

```
1  import { Dish } from "src/dishes/entities/dish.entity";
2  import { Column, Entity, JoinColumn,ManyToOne, PrimaryGeneratedColumn } from "typeorm";
3
4  @Entity({ name: "Images" })
5  export class Image {
6      @PrimaryGeneratedColumn({ name: "ImageID" })
7      ImageID: number
8
9
10     @Column({
11         type: "varchar",
12         name: "Path",
13         nullable: true
14     })
15     Path?: string
16
17     @Column({
18         type: "varchar",
19         name: "OriginalName",
20         nullable: true,
21     })
22     OriginalName?: string
23
24     @Column({
25         type: "varchar",
26         name: "UniqueName",
27         nullable: true
28     })
29     UniqueName?: string
30
31     @Column({
32         type: "date",
33         name: "CreationDate",
34         nullable: true,
35         default: () => "SYSDATETIME()"
36     })
37     CreationDate?: string
38
39
40     @Column({
41         type: "bit",
42         name: "IsActive",
43         nullable: true,
44         default: 1
45     })
46     IsActive?: number
47
48     @ManyToOne(() => Dish)
49     @JoinColumn({ name: "DishID" })
50     Dish: Dish
51 }
```

Figura 32. código image.entity

Clase image.controller:

```
1  import { Controller, UseInterceptors, Post, Get, UploadedFile, Body, Param, ParseIntPipe, Delete } from "@nestjs/common";
2  import { FileInterceptor } from "@nestjs/platform-express";
3  import { diskStorage } from "multer";
4  import { fileFilter, renameImage } from "../helpers/images.helper";
5  import { ImageServices } from "../services/image.service";
6  import { Image } from "../entities/image.entity";
7
8  @Controller("images")
9  export class ImageController {
10     constructor(private readonly imageService: ImageServices) { }
11
12     @Get()
13     GetImages(): Promise<Image[]> {
14         return this.imageService.GetImages();
15     }
16
17     @Get(":id")
18     GetImage(@Param("id", ParseIntPipe) id: number): Promise<Image> {
19         return this.imageService.GetImage(id)
20     }
21
22     @Get("dish/:id")
23     GetImageByDish(@Param("id", ParseIntPipe) id: number): Promise<Image[]> {
24         return this.imageService.GetImagesByDish(id);
25     }
26
27     @Post("upload")
28     @UseInterceptors(FileInterceptor("file", {
29         storage: diskStorage({
30             destination: "./uploads",
31             filename: renameImage
32         }),
33         fileFilter: fileFilter
34     }))
35     UploadImage(@UploadedFile() file: Express.Multer.File, @Body() body: any): Promise<Image> {
36         return this.imageService.SaveImage(body.DishID, file.originalname, file.filename, file.path)
37     }
38
39
40     @Delete(":id")
41     RemoveImage(@Param("id", ParseIntPipe) id: number) {
42         this.imageService.RemoveImage(id)
43     }
44 }
```

Figura 33. código image.controller

Class image.service:

```
1
2 import { Injectable, NotFoundException } from "@nestjs/common";
3 import { InjectRepository } from "@nestjs/typeorm";
4 import { Image } from "../entities/image.entity";
5 import { Repository } from "typeorm";
6 import { DishService } from "src/dishes/services/dish.service";
7 import { backgroundRemove } from "../helpers/background_remove.helper";
8
9 @Injectable()
10 export class ImageServices {
11
12   constructor(
13     @InjectRepository(Image)
14     private imageRepository: Repository<Image>,
15     private readonly dishService: DishService
16   ) { }
17
18   async GetImages(): Promise<Image[]> {
19     let images = await this.imageRepository.find({ where: { IsActive: 1 } })
20     return images
21   }
22
23   async GetImagesByDish(dishID: number): Promise<Image[]> {
24     let images = await this.imageRepository.find({ where: { IsActive: 1, Dish: { DishID: dishID } } })
25     return images
26   }
27
28   async GetImage(id: number): Promise<Image> {
29     let image = await this.imageRepository.findOne({ where: { IsActive: 1, ImageID: id } })
30     if (!image) throw new NotFoundException("No se encontro la imagen buscada")
31     return image
32   }
33
34   async SaveImage(dishID: number, originalname: string, filename: string, path: string): Promise<Image> {
35     let newImage = this.imageRepository.create({
36       OriginalName: originalname,
37       UniqueName: filename,
38       Path: `api/${path}`
39     })
40     let dish = await this.dishService.GetDish(dishID);
41     newImage.Dish = dish;
42     let [image] = await Promise.all([
43       this.imageRepository.save(newImage),
44       this.dishService.AddImage(newImage, dish.DishID),
45       backgroundRemove(path)
46     ])
47     return image;
48   }
49
50   async RemoveImage(id: number) {
51     let image = await this.GetImage(id);
52     image.IsActive = 0;
53     await this.imageRepository.save(image);
54   }
55 }
```

Figura 34. código image.service

Clase category.entity:

```
1  import { Dish } from "src/dishes/entities/dish.entity"
2  import { Entity, PrimaryGeneratedColumn, Column, ManyToOne, JoinColumn, OneToMany } from "typeorm"
3
4
5  @Entity({ name: "Categories" })
6  export class Category {
7      @PrimaryGeneratedColumn({ name: "CategoryID" })
8      CategoryID: number
9
10     @Column({
11         type: "varchar",
12         name: "CategoryName",
13         length: 150,
14         nullable: false,
15     })
16     CategoryName: string
17
18     @Column({
19         type: "bit",
20         name: "IsActive",
21         default: 1,
22         nullable: true
23     })
24     IsActive?: number
25
26     @Column({
27         type: "date",
28         name: "CreationDate",
29         default: () => "SYSDATETIME()",
30         nullable: true,
31     })
32     CreationDate?: string
33
34     @OneToMany(() => Dish, (dish) => dish.Categories, { nullable: true })
35     @JoinColumn({ name: "DishID" })
36     Dish?: Dish
37 }
```

Figura 35. código *category.entity*

Class category.controller:

```
1  import { Body, Controller, Delete, Get, Param, ParseIntPipe, Post } from "@nestjs/common";
2  import { CategoryService } from "../services/category.service";
3  import { Category } from "../entities/category.entity";
4  import { CreateCategoryDto } from "../dtos/create-category.dto";
5
6  @Controller("categories")
7  export class CategoryController {
8
9      constructor(private readonly categoryService: CategoryService) { }
10
11      // Get all categories
12      @Get()
13      GetCategories(): Promise<Category[]> {
14          return this.categoryService.GetCategories()
15      }
16
17      // Get a category by its id
18      @Get(":id")
19      GetCategory(@Param("id", ParseIntPipe) id: number): Promise<Category> {
20          return this.categoryService.GetCategory(id)
21      }
22
23      // Add a new category
24      @Post()
25      AddCategory(@Body() createCategoryDto: CreateCategoryDto): Promise<Category> {
26          return this.categoryService.AddCategory(createCategoryDto)
27      }
28
29      // Delete a category
30      @Delete(":id")
31      RemoveCategory(@Param("id", ParseIntPipe) id: number) {
32          this.categoryService.RemoveCategory(id)
33      }
34  }
```

Figura 36. código *category.controller*

Clase category.service:

```
1  import { Module } from "@nestjs/common";
2  import { TypeOrmModule } from "@nestjs/typeorm"
3  import { Category } from "../entities/category.entity";
4  import { CategoryController } from "../controllers/category.controller";
5  import { CategoryService } from "../services/category.service";
6  import { Dish } from "src/dishes/entities/dish.entity";
7
8  @Module({
9    imports: [TypeOrmModule.forFeature([Category, Dish])],
10   providers: [CategoryService],
11   controllers: [CategoryController],
12   exports: [CategoryService]
13 })
14 export class CategoryModule {
15
16 }
```

Figura 37. código *category.service*

CRUD ORDER:

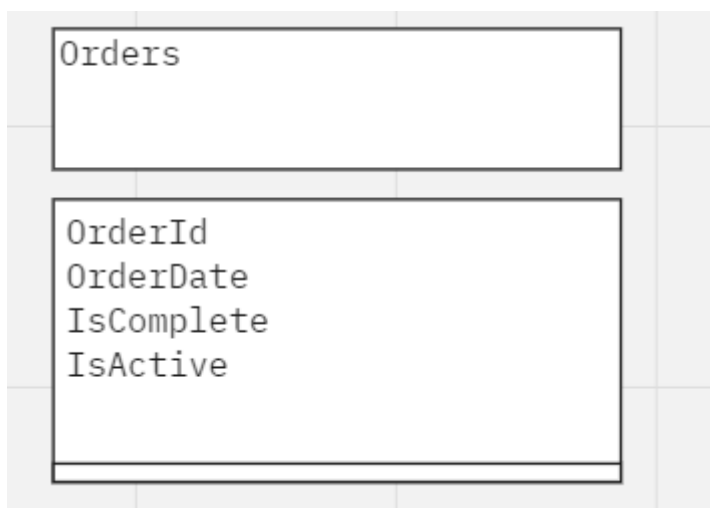


Figura 38. diagrama de entidades *Orders*

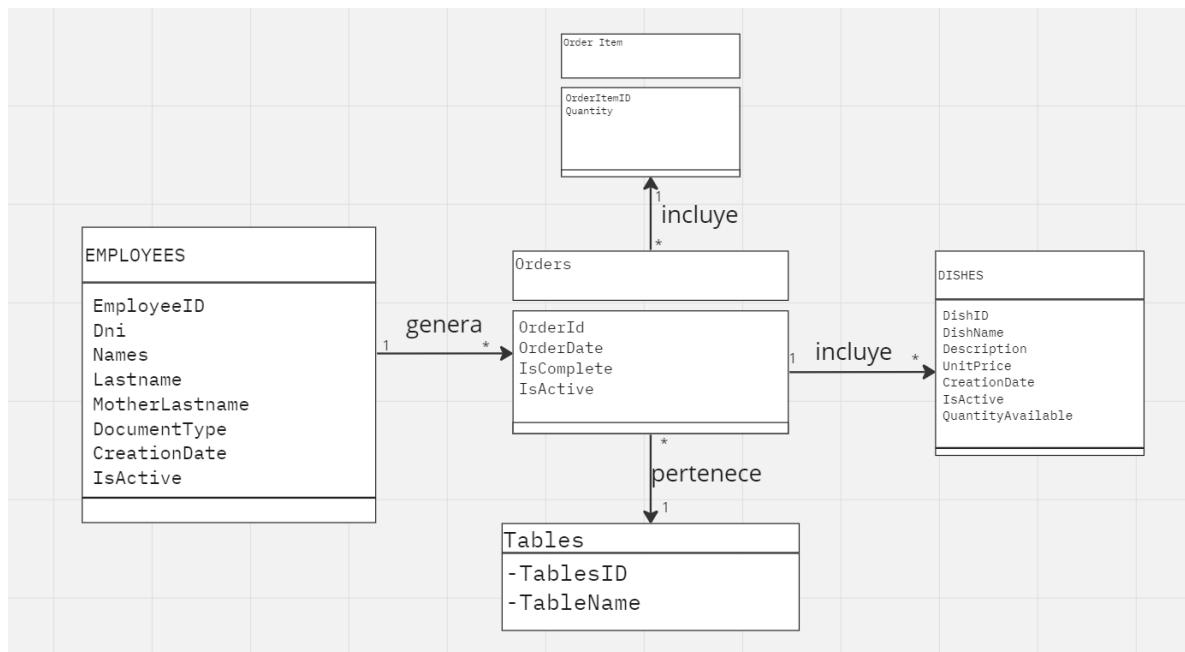


Figura 39. diagrama de entidades

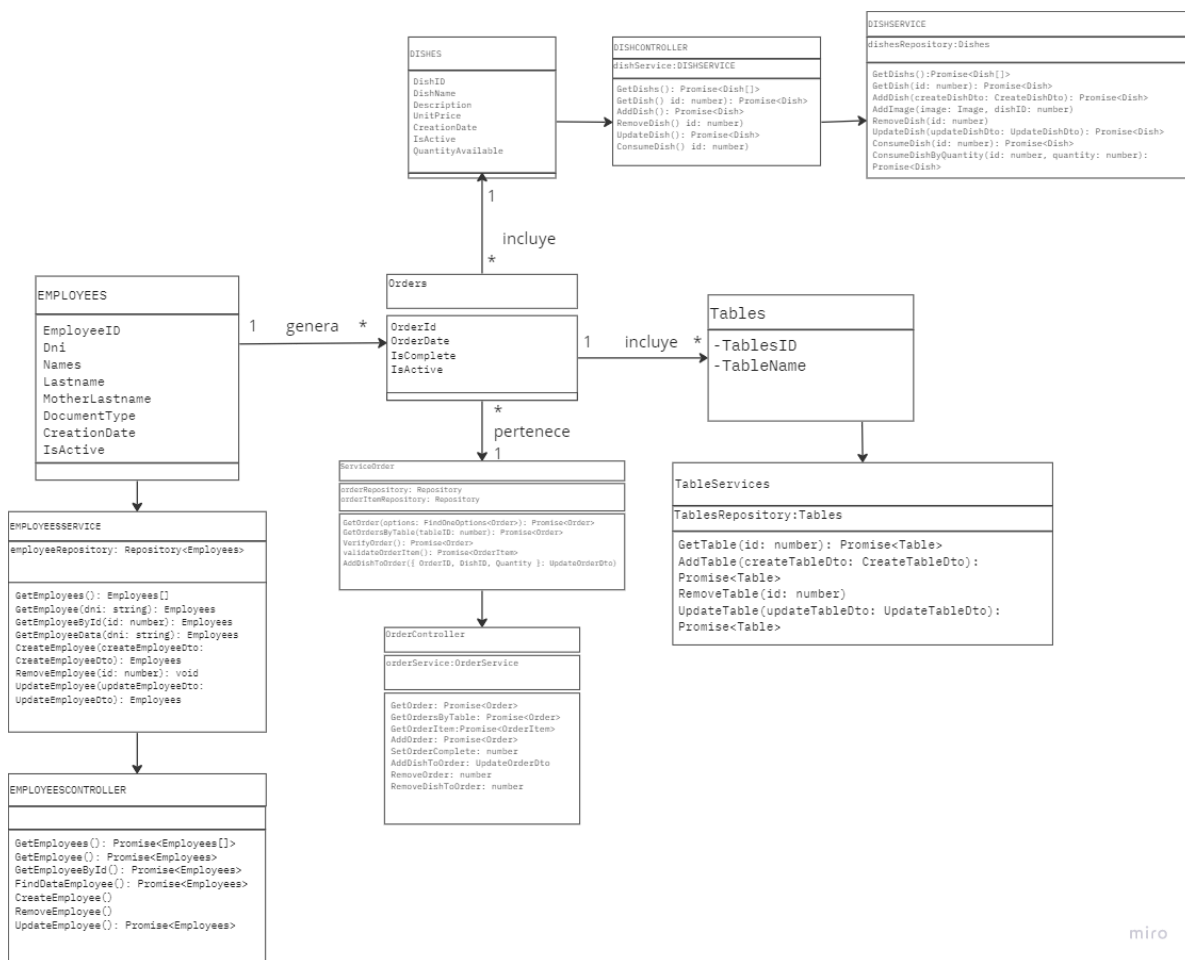


Figura 40. diagrama de entidades con cardinalidad

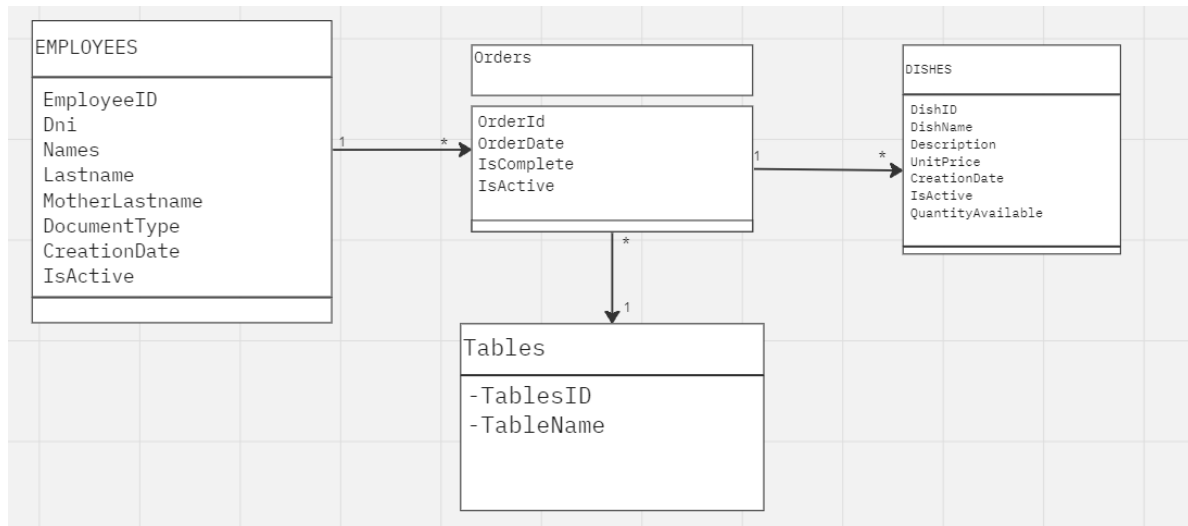


Figura 41. *diagrama de entidades con cardinalidad*

Clase orders. entity:

```

7  export class Order {
8      @PrimaryGeneratedColumn({ name: "OrderID" })
9      OrderID: number
10
11
12      @Column({
13          type: "date",
14          name: "OrderDate",
15          default: () => "SYSDATETIME()",
16          nullable: true
17      })
18      OrderDate?: string
19
20      @Column({
21          type: "bit",
22          name: "IsComplete",
23          default: 0,
24          nullable: true
25      })
26      IsComplete?: number
27
28      @Column({
29          type: "bit",
30          name: "IsActive",
31          default: 1,
32          nullable: true
33      })
34      IsActive?: number
35
36      @ManyToOne(() => Table, { eager: true })
37      @JoinColumn({ name: "TableID" })
38      Table: Table
39
40      @OneToMany(() => OrderItem, (orderItem) => orderItem.Order, { eager: true })
41      OrderItems: OrderItem[];
42
43      @ManyToOne(() => Employees, { eager: true })
44      @JoinColumn({ name: "EmployeeID" })
45      Employee: Employees
46  }
  
```

Figura 42. *código orders.entity*

OrderItem:

```
49  export class OrderItem {
50      @PrimaryGeneratedColumn()
51      OrderItemID: number
52
53      @ManyToOne(() => Order, (order) => order.OrderItems)
54      @JoinColumn({ name: "OrderID" })
55      Order: Order
56
57      @ManyToOne(() => Dish, { eager: true })
58      @JoinColumn({ name: "DishID" })
59      Dish: Dish
60
61      @Column({
62          type: "tinyint",
63          name: "Quantity",
64          nullable: false
65      })
66      Quantity: number
67  }
```

Figura 43. código *ordersItem*
Clase *order.controller*:

```

7   @Controller("orders")
8   export class OrderController {
9       constructor(private readonly orderService: OrderService) { }
10
11       @Get()
12       GetOrders() {
13           return this.orderService.GetOrders()
14       }
15
16       @Get("/:id")
17       GetOrder(@Param("id", ParseIntPipe) id: number): Promise<Order> {
18           return this.orderService.GetOrder({ where: { OrderID: id, IsComplete: 0 } })
19       }
20
21       @Get("table/:id")
22       GetOrdersByTable(@Param("id", ParseIntPipe) id: number): Promise<Order> {
23           return this.orderService.GetOrdersByTable(id);
24       }
25
26       @Get("item/:id")
27       GetOrderItem(@Param("id", ParseIntPipe) id: number): Promise<OrderItem> {
28           return this.orderService.GetOrderItem(id)
29       }
30
31       @Post()
32       AddOrder(@Body() createOrderDto: CreateOrderDto): Promise<Order> {
33           return this.orderService.AddOrder(createOrderDto)
34       }
35
36       @Put("/:id")
37       SetOrderComplete(@Param("id", ParseIntPipe) id: number) {
38           return this.orderService.SetOrderComplete(id)
39       }
40
41       @Put()
42       AddDishToOrder(@Body() updateOrderDto: UpdateOrderDto) {
43           this.orderService.AddDishToOrder(updateOrderDto)
44       }
45
46       @Delete("/:id")
47       RemoveOrder(@Param("id", ParseIntPipe) id: number) {
48           this.orderService.RemoveOrder(id)
49       }
50
51       @Delete("item/:id")
52       RemoveDishToOrder(@Param("id", ParseIntPipe) id: number) {
53           this.orderService.RemoveDishToOrder(id)
54       }
55   }
56

```

Figura 43. código *order.controller*
Clase *order.service*:

Code Blame 130 lines (115 loc) · 5.18 KB

```

13 export class OrderService {
14
15   constructor(
16     @InjectRepository(Order) private orderRepository: Repository<Order>,
17     @InjectRepository(OrderItem) private orderItemRepository: Repository<OrderItem>,
18     private readonly tableService: TableService,
19     private readonly dishService: DishService,
20     private readonly employeeService: EmployeeService
21   ) { }
22
23
24   async GetOrders() {
25     let orders = await this.orderRepository.find({ where: { IsActive: 1 } })
26     return orders
27   }
28
29   async GetOrder(options: FindOneOptions<Order>): Promise<Order> {
30     let order: Order = await this.orderRepository.findOne(options);
31     if (!order)
32       throw new NotFoundException("No se encontro ninguna orden con el id: '${options.order.OrderID}'")
33     return order
34   }
35
36   async GetOrdersByTable(tableID: number): Promise<Order> {
37     let order = await this.orderRepository.findOne({ where: { Table: { TableID: tableID }, IsActive: 1, IsComplete: 0 } })
38     if (!order) throw new NotFoundException("No se encontro ordenes para esta mesa");
39     return order;
40   }
41
42   async VerifyOrder(tableID: number, orderDishes: OrderDish[], employeeID: number): Promise<Order> {
43     let order = await this.orderRepository.findOne({ where: { Table: { TableID: tableID }, IsComplete: 0 } })
44     if (!order) {
45       let [table, employee] = await Promise.all([
46         this.tableService.GetTable(tableID),
47         this.employeeService.GetEmployeeById(employeeID)
48       ]);
49       let availableDishes = []
50       let newOrder = this.orderRepository.create({ Table: table, Employee: employee })
51       for (let orderDish of orderDishes) {
52         let orderItem = await Promise.all([
53           this.validateOrderItem(orderDish.DishID, orderDish.Quantity),
54           this.employeeService.GetEmployeeById(employeeID)
55         ]);
56         availableDishes.push(orderItem)
57         newOrder.OrderItems = availableDishes
58       }
59       return newOrder
60     }
61     else {
62       for (let orderDish of orderDishes) {
63         let orderItem = await this.validateOrderItem(orderDish.DishID, orderDish.Quantity)
64         order.OrderItems.push(orderItem)
65       }
66       return order
67     }
68   }

```

Figura 44. código order.service
CLASE dish.entity:

```

7  ✓ export class Dish {
8      @PrimaryGeneratedColumn({ name: "DishID" })
9      DishID: number
10
11      @Column({
12          type: "varchar",
13          name: "DishName",
14          length: 150,
15          nullable: false
16      })
17      @Index({ unique: false })
18      DishName: string
19
20      @Column({
21          type: "text",
22          name: "Description",
23          nullable: true
24      })
25      Description?: string
26
27      @Column({
28          name: "UnitPrice",
29          type: "money",
30          nullable: false
31      })
32      UnitPrice: number
33
34      @Column({
35          name: "CreationDate",
36          type: "date",
37          nullable: true,
38          default: () => "SYSDATETIME()",
39      })
40      CreationDate?: string
41
42      @Column({
43          name: "IsActive",
44          type: "bit",
45          nullable: true,
46          default: 1,
47      })
48      IsActive?: number
49
50
51      @OneToMany(() => Image, (image) => image.Dish, { nullable: true, eager: true })
52      Images?: Image[]
53
54      @Column({
55          name: "QuantityAvailable",
56          type: "int",
57          nullable: true,
58          default: 0
59      })
60      QuantityAvailable?: number
61

```

Figura 45. código dish.entity

Clase dish.controller:


```

import { Body, Controller, Delete, Get, Param, ParseIntPipe, Post, Put } from "@nestjs/common";
import { DishService } from "../services/dish.service";
import { Dish } from "../entities/dish.entity";
import { CreateDishDto } from "../dtos/create-dish.dto";
import { UpdateDishDto } from "../dtos/update-dish.dto";

@Controller("dishes")
✓ export class DishController {
  constructor(private readonly dishService: DishService) { }

  @Get()
  GetDishes(): Promise<Dish[]> {
    return this.dishService.GetDishes()
  }

  @Get(":id")
  GetDish(@Param("id", ParseIntPipe) id: number): Promise<Dish> {
    return this.dishService.GetDish(id)
  }

  @Get("search/category/:category")
  GetDishesByCategory(@Param("category") category: string): Promise<Dish[]> {
    return this.dishService.GetDishesByCategory(category);
  }

  @Get("search/name/:name")
  GetDishesByName(@Param("name") name: string): Promise<Dish[]> {
    return this.dishService.GetDishesByName(name);
  }

  @Post()
  AddDish(@Body() createDishDto: CreateDishDto): Promise<Dish> {
    return this.dishService.AddDish(createDishDto)
  }

  @Delete(":id")
  RemoveDish(@Param("id", ParseIntPipe) id: number) {
    this.dishService.RemoveDish(id)
  }

  @Put()
  UpdateDish(@Body() updateDishDto: UpdateDishDto): Promise<Dish> {
    return this.dishService.UpdateDish(updateDishDto);
  }

  @Put(":id")
  ConsumeDish(@Param("id", ParseIntPipe) id: number) {
    this.dishService.ConsumeDish(id)
  }
}

```

Figura 46. código *dish.controller*
Clase *dish.service*:

```

@Injectables()
export class DishService {
  constructor(
    @InjectRepository(Dish)
    private dishRepository: Repository<Dish>,
    private readonly categoryService: CategoryService
  ) { }

  async GetDishes(): Promise<Dish[]> {
    let dishes = await this.dishRepository.find({
      where: { IsActive: 1 }, relations: { Categories: true }
    })
    return dishes
  }

  async GetDish(id: number): Promise<Dish> {
    let dish = await this.dishRepository.findOne({ where: { DishID: id, IsActive: 1 } })
    if (!dish) throw new NotFoundException('No se encontro el platillo con el id: '${id}'')
    return dish
  }

  async GetDishesByCategory(category: string): Promise<Dish[]> {
    let dishes = await this.dishRepository.find({ where: { Categories: { CategoryName: category }, IsActive: 1 } })
    if (dishes.length === 0) throw new NotFoundException('No se encontraron platillos con esa categoria')
    return dishes;
  }

  async GetDishesByName(name: string): Promise<Dish[]> {
    let dishes = await this.dishRepository.find({ where: { IsActive: 1, DishName: Like(`%${name}%`) } })
    if (dishes.length === 0) throw new NotFoundException('No se encontraron platillos con dicho nombre')
    return dishes;
  }

  async AddDish(createDishDto: CreateDishDto): Promise<Dish> {
    let categories = []
    if (createDishDto.CategoriesID) {
      categories = createDishDto.CategoriesID.map((categoryID) => this.categoryService.GetCategory(categoryID));
      console.log(categories);
    }
    let newDish = this.dishRepository.create(createDishDto)
    newDish.Categories = await Promise.all(categories)
    return await this.dishRepository.save(newDish)
  }

  async AddImage(image: Image, dishID: number) {
    let dish = await this.GetDish(dishID);
    if (!dish.Images) dish.Images = []
    dish.Images.push(image);
    await this.dishRepository.save(dish);
  }
}

```

```

async RemoveDish(id: number) {
    let dish = await this.GetDish(id)
    dish.IsActive = 0
    await this.dishRepository.save(dish)
}

async UpdateDish(updateDishDto: UpdateDishDto): Promise<Dish> {
    let dish = await this.GetDish(updateDishDto.DishID)
    Object.assign(dish, updateDishDto)
    return await this.dishRepository.save(dish)
}

async ConsumeDish(id: number): Promise<Dish> {
    let dish = await this.GetDish(id)
    if (dish.QuantityAvailable != null && dish.QuantityAvailable > 0) {
        --dish.QuantityAvailable
        return await this.dishRepository.save(dish)
    }
    return null
}

async ConsumeDishByQuantity(id: number, quantity: number): Promise<Dish> {
    let dish = await this.GetDish(id)
    if (dish.QuantityAvailable != null && dish.QuantityAvailable > 0 && quantity <= dish.QuantityAvailable) {
        dish.QuantityAvailable -= quantity
        return await this.dishRepository.save(dish)
    }
    return null
}

```

Figura 47. código *dish.service*
Clase employees.entity:

```

5    @Entity({ name: "Employees" })
6  ✓ export class Employees {
7
8      @PrimaryGeneratedColumn()
9      EmployeeID: number
10
11      @Column({
12          type: "char",
13          length: 8,
14          nullable: false
15      })
16      @Index({ unique: true })
17      Dni: string
18
19      @Column({
20          type: "varchar",
21          name: "Names",
22          nullable: false
23      })
24      Names: string;
25
26      @Column({
27          type: "varchar",
28          name: "Lastname",
29          nullable: false
30      })
31      Lastname: string;
32
33      @Column({
34          type: "varchar",
35          name: "MotherLastname",
36          nullable: false
37      })
38      MotherLastname: string
39
40      @Column({
41          type: "varchar",
42          name: "DocumentType",
43          nullable: false,
44          length: 1
45      })
46      DocumentType: string
47
48      @Column({
49          type: "date",
50          name: "CreationDate",
51          nullable: true,
52          default: () => "SYSDATETIME()"
53      })
54      CreationDate?: string;

```

```
56     @Column({
57         type: "bit",
58         name: "IsActive",
59         nullable: true,
60         default: 1
61     })
62     IsActive?: number;
63
64     @OneToMany(() => Order, (order) => order.Employee)
65     Orders: Order[]
66
67     @OneToMany(() => Message, (message) => message.Employee)
68     Messages: Message[]
69 }
```

Figura 48. *código employees.entity*

Clase EmployeesController:

IngSoft-CartillaVirtual / backend / src / employees / controllers / employees.controller.ts 

 Apo-Theddy Se agregaron nuevas funcionalidades al backend y al aplicativo de tel... 

Code

Blame

45 lines (37 loc) · 1.48 KB

```
1 import { Controller, Delete, Get, Param, Post, ParseIntPipe, Put, Body } from "@nestjs/common";
2 import { EmployeeService } from "../service/employees.service";
3 import { Employees } from "../entities/employees.entity";
4 import { UpdateEmployeeDto } from "../dtos/update-employee.dto";
5 import { CreateEmployeeDto } from "../dtos/create-employee.dto";
6
7 @Controller("employees")
8 export class EmployeesController {
9     constructor(private readonly employeeService: EmployeeService) {}
10
11     @Get()
12     GetEmployees(): Promise<Employees[]> {
13         return this.employeeService.GetEmployees();
14     }
15
16     @Get("/:dni")
17     GetEmployee(@Param("dni") dni: string): Promise<Employees> {
18         return this.employeeService.GetEmployeeData(dni)
19     }
20
21     @Get("search/:id")
22     GetEmployeeById(@Param("id", ParseIntPipe) id: number): Promise<Employees> {
23         return this.employeeService.GetEmployeeById(id);
24     }
25
26     @Post("/:dni")
27     FindDataEmployee(@Param("dni",) dni: string): Promise<Employees> {
28         return this.employeeService.FindDataEmployee(dni);
29     }
30
31     @Post()
32     CreateEmployee(@Body() createEmployeeDto: CreateEmployeeDto) {
33         return this.employeeService.CreateEmployee(createEmployeeDto);
34     }
35
36     @Delete("/:id")
37     RemoveEmployee(@Param("id",) id: number) {
38         this.employeeService.RemoveEmployee(id)
39     }
40
41     @Put()
42     UpdateEmployee(@Body() updateEmployeeDto: UpdateEmployeeDto): Promise<Employees> {
43         return this.employeeService.UpdateEmployee(updateEmployeeDto);
44     }
45 }
```

Figura 49. código employees.controller

Clase EmployeeService:

IngSoft-CartillaVirtual / backend / src / employees / service / employees.service.ts

Code

Blame

75 lines (63 loc) · 3.03 KB

```
12 export class EmployeeService {
16     private readonly reniecService: ApiReniecDataService,
17 } { }
18
19
20 async GetEmployees(): Promise<Employees[]> {
21     let employees = await this.employeeRepository.find({ where: { IsActive: 1 } });
22     return employees;
23 }
24
25 async GetEmployee(dni: string): Promise<Employees> {
26     let employee = await this.employeeRepository.findOne({ where: { Dni: dni, IsActive: 1 } })
27     return employee;
28 }
29
30 ✓ async GetEmployeeById(id: number): Promise<Employees> {
31     let employee = await this.employeeRepository.findOne({ where: { EmployeeID: id, IsActive: 1 } })
32     if (!employee) throw new NotFoundException(`No se encontro el empleado con el id: ${id}`)
33     return employee;
34 }
35
36 ✓ async GetEmployeeData(dni: string): Promise<Employees> {
37     let employee = await this.GetEmployee(dni)
38     if (!employee) throw new NotFoundException(`No se encontro al empleado con el dni: ${dni}`);
39     return employee;
40 }
41
42 ✓ async FindDataEmployee(dni: string): Promise<Employees> {
43     try {
44         let employee = await this.GetEmployee(dni);
45         if (!employee) {
46             let newEmployee = this.employeeRepository.create(await this.reniecService.findUserByDni(dni));
47             return await this.employeeRepository.save(newEmployee)
48         }
49         throw new ConflictException(`El empleado con el dni: ${dni} ya se encuentra registrado`)
50     }
51     catch (err) {
52         throw new ConflictException(err);
53     }
54 }
55
56 ✓ async CreateEmployee(createEmployeeDto: CreateEmployeeDto): Promise<Employees> {
57     let employee = await this.GetEmployee(createEmployeeDto.Dni);
58     if (employee) throw new ConflictException(`El dni: ${createEmployeeDto.Dni} ya se encuentra registrado`)
59     let newEmployee = this.employeeRepository.create(createEmployeeDto);
60     return await this.employeeRepository.save(newEmployee);
61 }
62
63 ✓ async RemoveEmployee(id: number) {
64     let employee = await this.GetEmployeeById(id)
65     employee.IsActive = 0
66     await this.employeeRepository.save(employee);
67 }
```

```

68
69  ✓    async UpdateEmployee(updateEmployeeDto: UpdateEmployeeDto): Promise<Employees> {
70      let employee = await this.GetEmployeeData(updateEmployeeDto.Dni)
71      Object.assign(employee, updateEmployeeDto);
72      return await this.employeeRepository.save(employee);
73    }
74
75  }

```

Figura 50. código *employees.service*

Clase Table:

[IngSoft-CartillaVirtual](#) / [backend](#) / [src](#) / [tables](#) / [entities](#) / [table.entity.ts](#) 

 Apo-Theddy Agregando funciones al backend y conectando la aplicacion con el backend

Code Blame 36 lines (30 loc) • 784 Bytes


```


1  import { Order } from "src/orders/entities/order.entity"
2  import { Entity, PrimaryGeneratedColumn, Column, JoinColumn, OneToMany, OneToOne } from "typeorm"
3
4  @Entity({ name: "Tables" })
5  ✓ export class Table {
6
7      @PrimaryGeneratedColumn({ name: "TableID" })
8      TableID: number
9
10     @Column({
11         type: "varchar",
12         name: "TableName",
13         length: 100,
14         nullable: false
15     })
16     TableName: string
17
18     @Column({
19         type: "date",
20         name: "CreationDate",
21         default: () => "SYSDATETIME()",
22         nullable: true
23     })
24     CreationDate?: string
25
26     @Column({
27         type: "bit",
28         name: "IsActive",
29         default: 1,
30         nullable: true
31     })
32     IsActive?: number
33
34     @OneToMany(() => Order, (order) => order.Table)
35     Order: Order
36 }

```

Figura 51. código *table*

Clase TableController:

[IngSoft-CartillaVirtual](#) / [backend](#) / [src](#) / [tables](#) / [controllers](#) / [table.controller.ts](#) 

 Apo-Theddy Agregando el backend y el aplicativo movil

```
Code Blame 36 lines (29 loc) · 1.07 KB

1  import { Body, Controller, Delete, Get, Param, ParseIntPipe, Post, Put } from "@nestjs/common";
2  import { Table } from "../entities/table.entity";
3  import { CreateTableDto } from "../dtos/create-table.dto";
4  import { UpdateTableDto } from "../dtos/update-table.dto";
5  import { TableService } from "../services/table.service";
6
7  @Controller("tables")
8  export class TableController {
9
10     constructor(private readonly tableService: TableService) { }
11
12     @Get()
13     GetTables(): Promise<Table[]> {
14         return this.tableService.GetTables()
15     }
16
17     @Get(":id")
18     GetTable(@Param("id", ParseIntPipe) id: number): Promise<Table> {
19         return this.tableService.GetTable(id)
20     }
21
22     @Post()
23     AddTable(@Body() createTableDto: CreateTableDto): Promise<Table> {
24         return this.tableService.AddTable(createTableDto)
25     }
26
27     @Delete(":id")
28     RemoveTable(@Param("id") id: number) {
29         this.tableService.RemoveTable(id)
30     }
31
32     @Put()
33     UpdateTable(@Body() updateTableDto: UpdateTableDto): Promise<Table> {
34         return this.tableService.UpdateTable(updateTableDto)
35     }
36 }
```

Figura 52. código *employees.controller*

Clase TableService:

```
1  import { Injectable, NotFoundException } from "@nestjs/common";
2  import { InjectRepository } from "@nestjs/typeorm"
3  import { Repository } from "typeorm"
4  import { Table } from "../entities/table.entity";
5  import { CreateTableDto } from "../dtos/create-table.dto";
6  import { UpdateTableDto } from "../dtos/update-table.dto";
7  import { DishService } from "src/dishes/services/dish.service";
8
9  @Injectable()
10 export class TableService {
11     constructor(
12         @InjectRepository(Table) private tableRepository: Repository<Table>
13         , private readonly dishService: DishService
14     ) { }
15
16     async GetTables(): Promise<Table[]> {
17         let tables = await this.tableRepository.find({ where: { IsActive: 1 } })
18         return tables
19     }
20
21     async GetTable(id: number): Promise<Table> {
22         let table = await this.tableRepository.findOne({ where: { TableID: id, IsActive: 1 } })
23         if (!table) throw new NotFoundException(`No se encontro la mesa con el id: '${id}'`)
24         return table
25     }
26
27     async AddTable(createTableDto: CreateTableDto): Promise<Table> {
28         let newTable = this.tableRepository.create(createTableDto)
29         return await this.tableRepository.save(newTable)
30     }
31
32     async RemoveTable(id: number) {
33         let table = await this.GetTable(id)
34         table.IsActive = 0
35         await this.tableRepository.save(table)
36     }
37
38     async UpdateTable(updateTableDto: UpdateTableDto): Promise<Table> {
39         let table = await this.GetTable(updateTableDto.TableID)
40         Object.assign(table, updateTableDto)
41         return await this.tableRepository.save(table)
42     }
43 }
```

Figura 53. *código table.service*
Incremento 02:

CRUD EMPLOYEES:

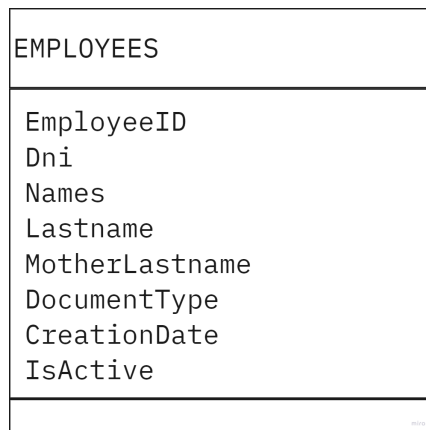


Figura 54. diagrama de entidades Employees

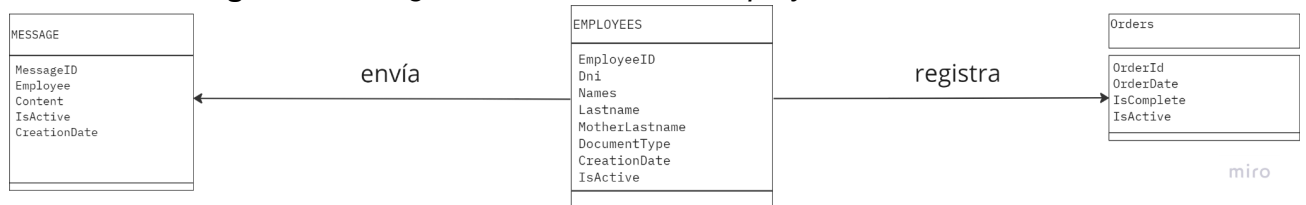


Figura 55. diagrama de entidades

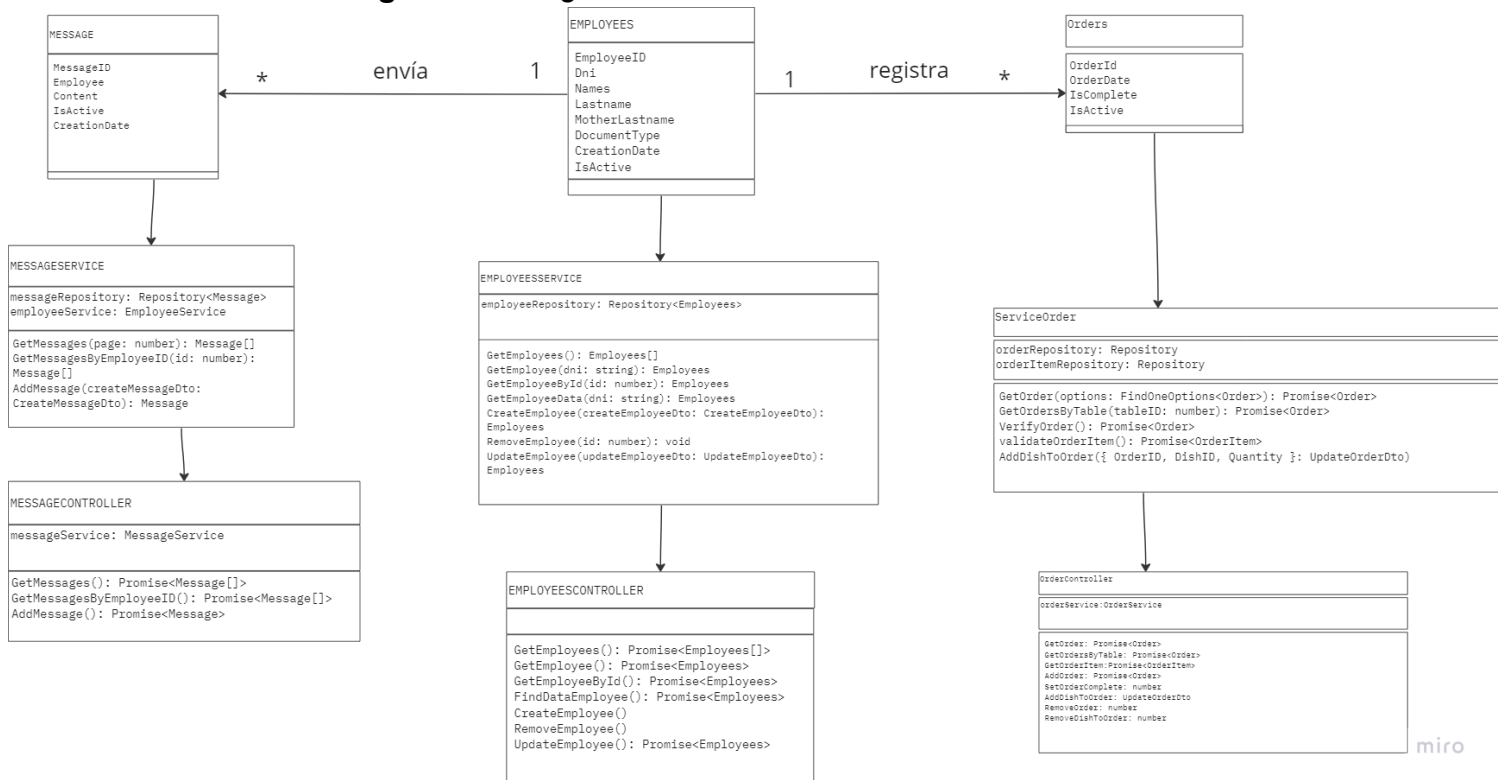


Figura 56. diagrama de entidades con cardinalidad

CRUD Tables:

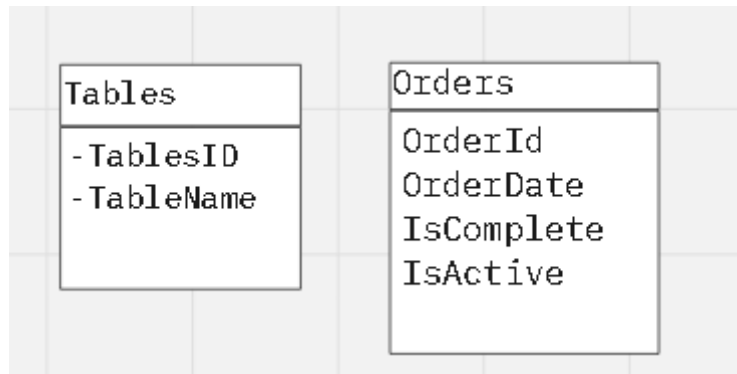


Figura 57. diagrama de entidades Tables

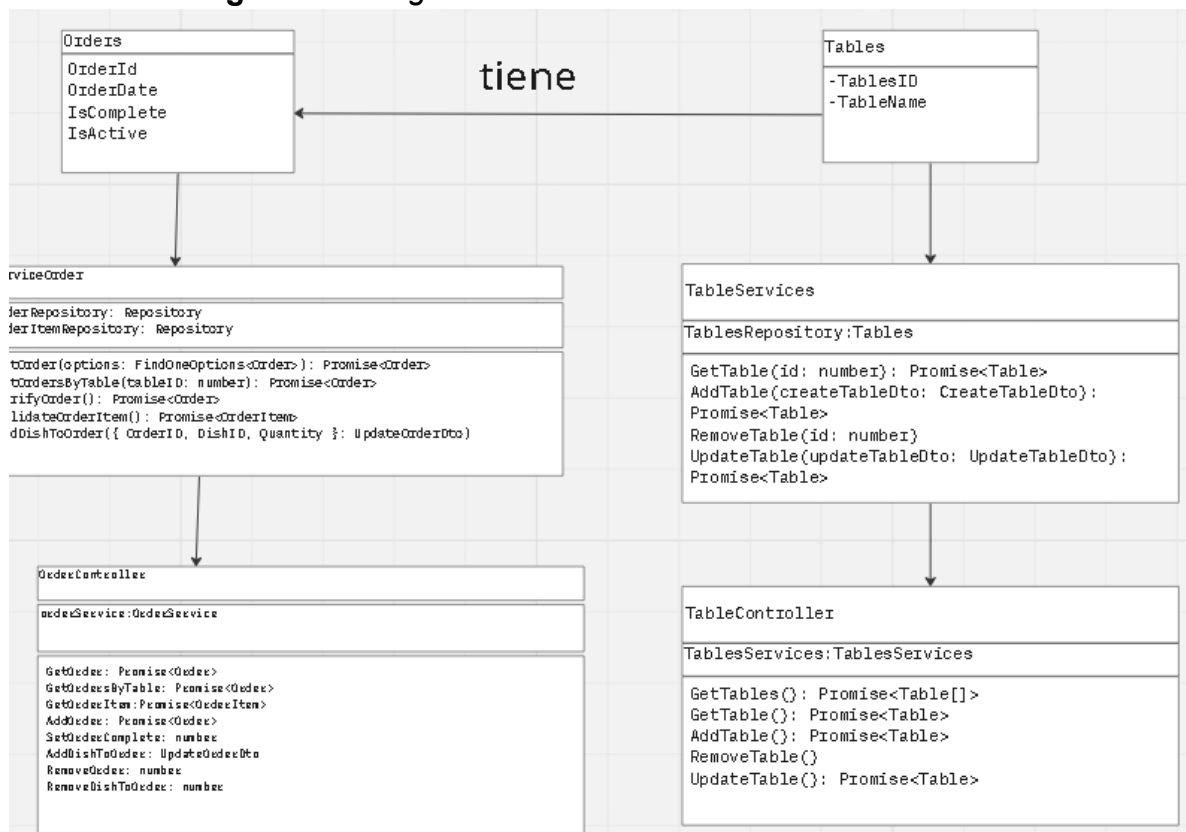
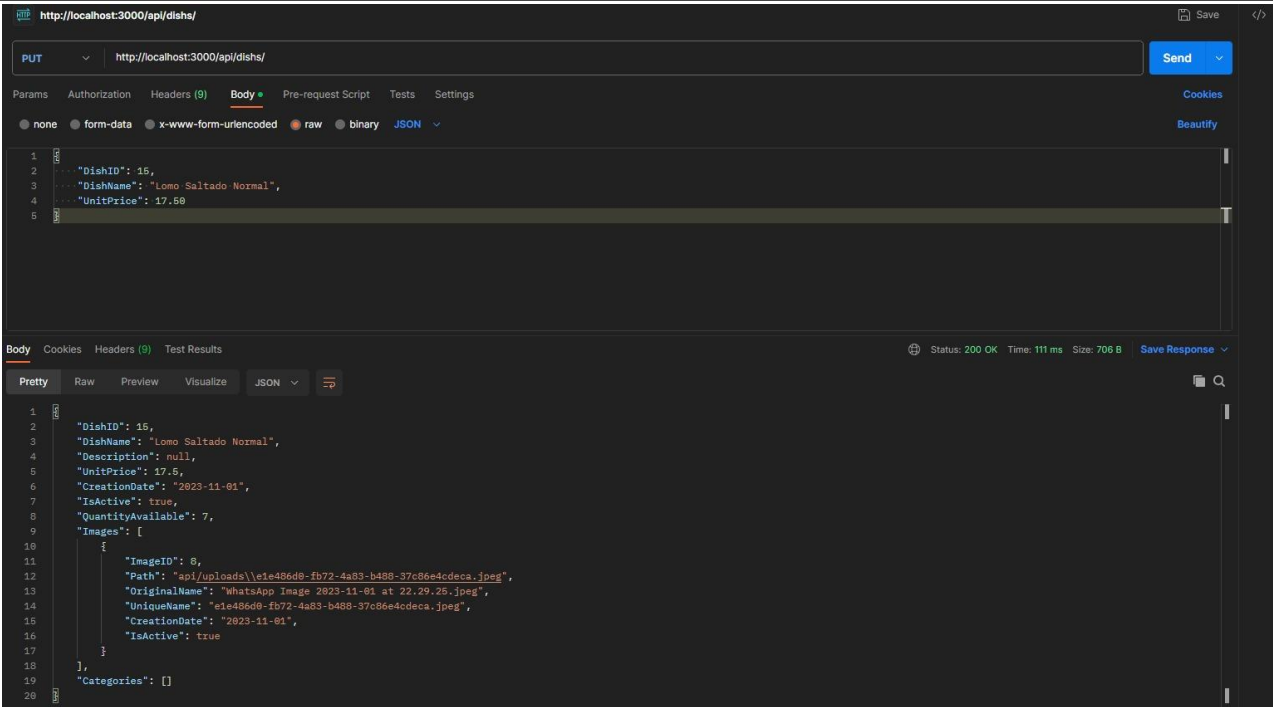
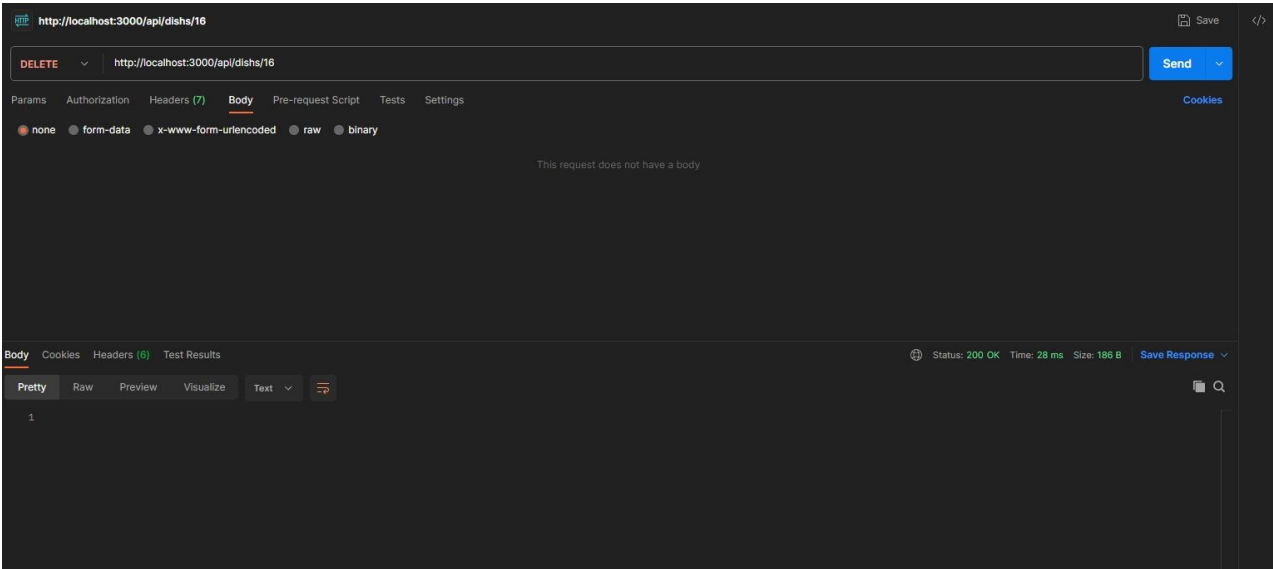
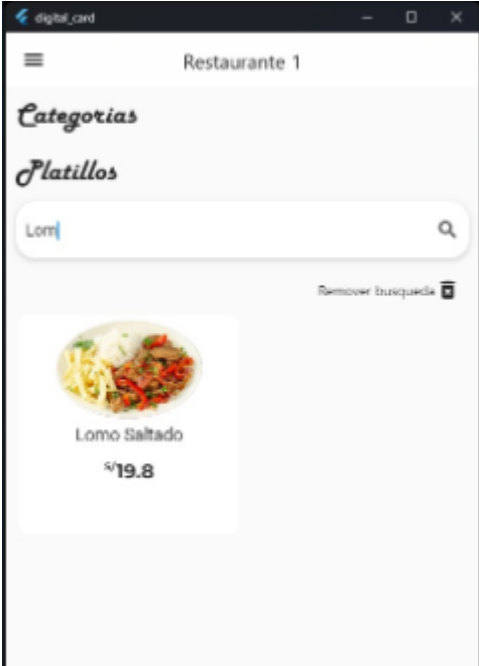


Figura 58. diagrama de entidades

ID	Caso de uso	Datos de entrada normal	Resultado esperado normal	Datos de entrada anómalo	Resultado esperado anómalo
1	Registrar Orden	OrderItemsIDs TableID EmployeeID	Registro ingresado satisfactorio	Nombre en blanco o unidad en blanco	Falta ingresar datos de nombre o unidad
Prueba					
2	Actualizar Orden	OrderID Quantity	Orden actualizada correctamente	Cantidad negativa	No se puede almacenar valores negativos
Prueba					

3	Eliminar Orden	IdOrden	Orden eliminada correctamente	Código no existente de orden	Fallo al eliminar orden
Prueba					
4	Registrar Platillos	DishName Description UnitPrice QuantityAvailable CategoryIDs	Platillo agregado correctamente	Nombre del plato vacío	Fallo al registrar el platillo
Prueba					
5	Actualizar Platillos	DishID Description UnitPrice QuantityAvailable CategoryIDs	Platillo actualizado correctamente	DishID no existente	Fallo al actualizar orden

Prueba					
6	Eliminar Platillos	DishId	Platillo eliminado correctamente	DishID no existente	Fallo al eliminar platillo
Prueba					
7	Buscar platillo	DishID	El platillo se encontro	DishID no existente	No se encontró el platillo

Prueba			
--------	--	--	--

ID	Caso de uso	Datos de entrada normal	Resultado esperado normal	Datos de entrada anómalo	Resultado esperado anómalo	Ciclo 1
1	Registrar Orden	OrderItemsIDs TableID EmployeeID	Registro ingresado satisfactorio	Nombre en blanco o unidad en blanco	Falta ingresar datos de nombre o unidad	Ok
2	Actualizar Orden	OrderID Quantity	Orden actualizada correctamente	Cantidad negativa	No se puede almacenar valores negativos	Ok
3	Eliminar Orden	IdOrden	Orden eliminada correctamente	Código no existente de orden	Fallo al eliminar orden	Ok
4	Registrar Platos	DishName Description UnitPrice QuantityAvailable CategoryIDs	Platillo agregado correctamente	Nombre del plato vacío	Fallo al registrar el platillo	Ok
5	Actualizar Platos	DishID Description UnitPrice QuantityAvailable CategoryIDs	Platillo actualizado correctamente	DishID no existente	Fallo al actualizar orden	Ok
6	Eliminar Platos	DishID	Platillo eliminado correctamente	DishID no existente	Fallo al eliminar platillo	Ok
7	Buscar platillo	DishID	El platillo se encontro	DishID no existente	No se encontró el platillo	ok

DIAGRAMA DE PAQUETES DE ARQUITECTURA

REGISTRAR MESA

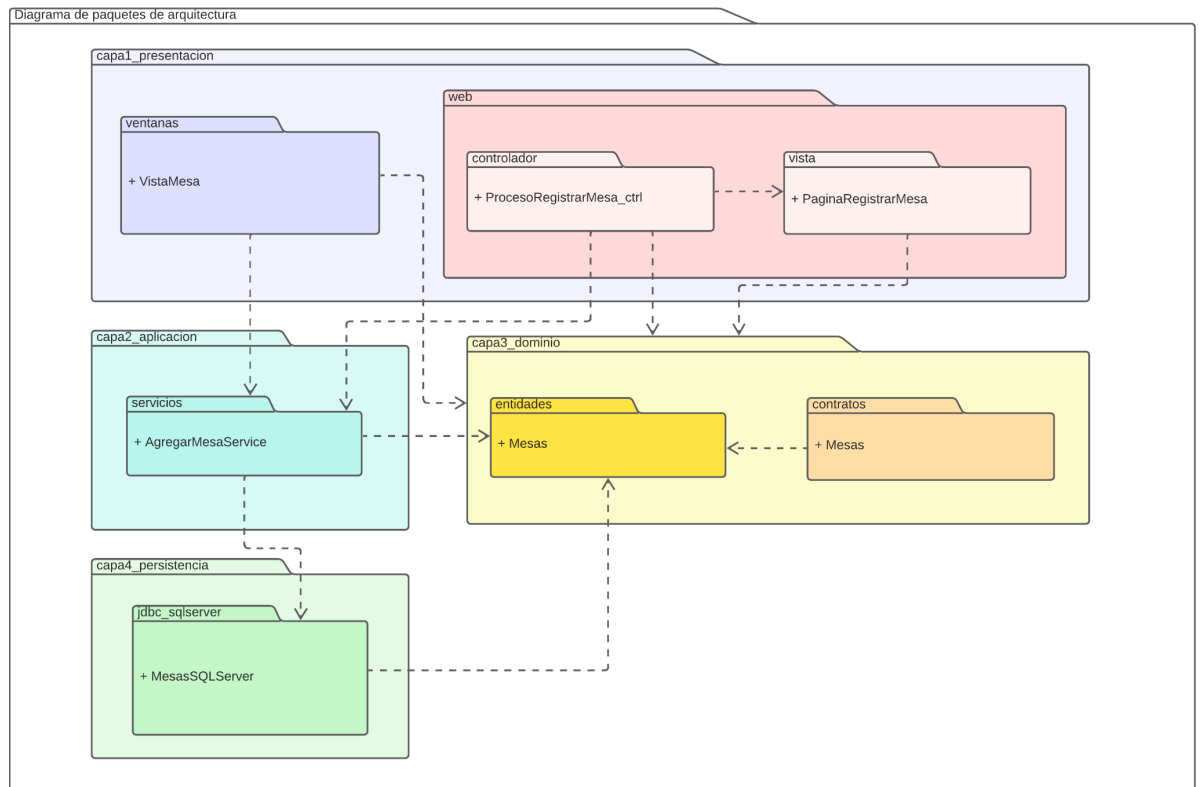


Figura 59. *diagrama de paquetes Registrar Mesa*

ACTUALIZAR MESA

Diagrama de paquetes de arquitectura

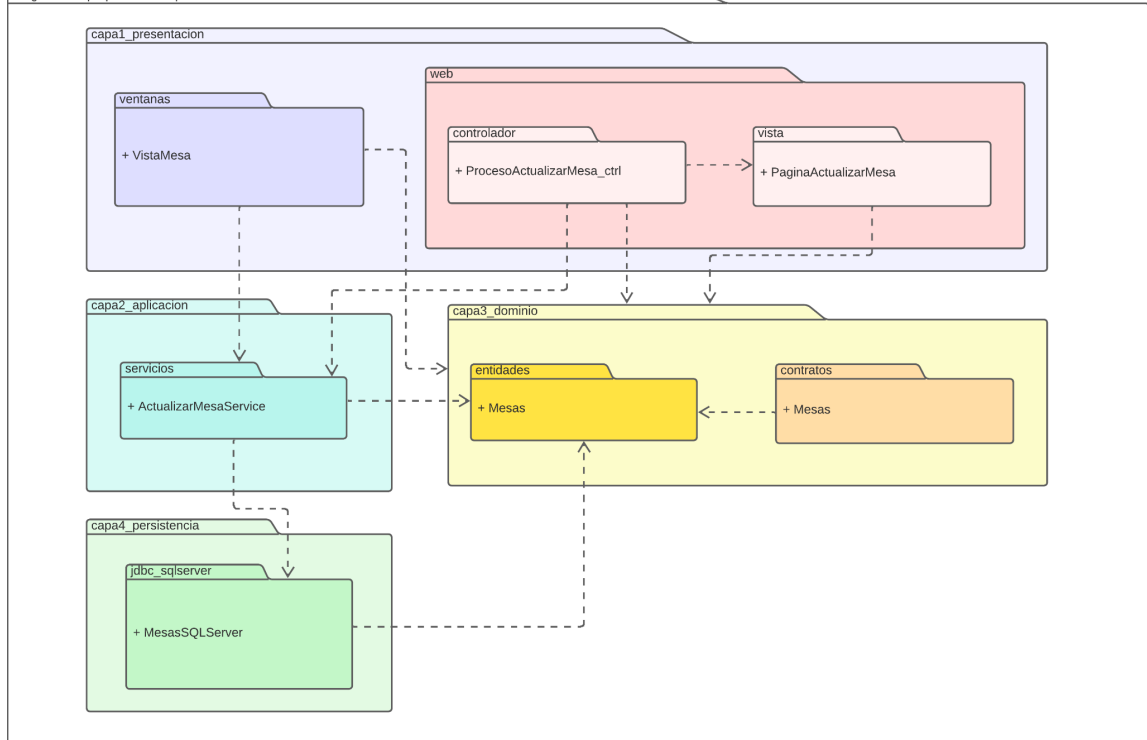


Figura 60. *diagrama de paquetes Actualizar Mesa*

ELIMINAR EMPLEADO

Diagrama de paquetes de arquitectura

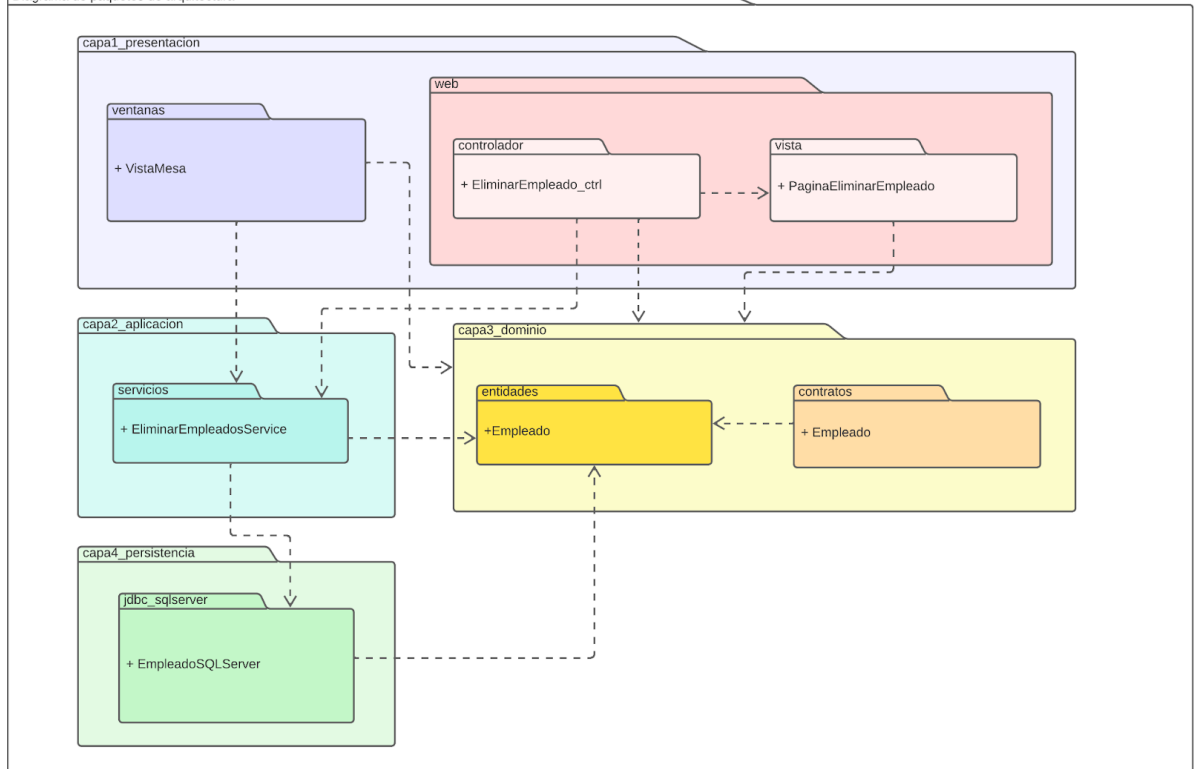


Figura 61. *diagrama de paquetes Eliminar Empleado*

REGISTRAR EMPLEADO

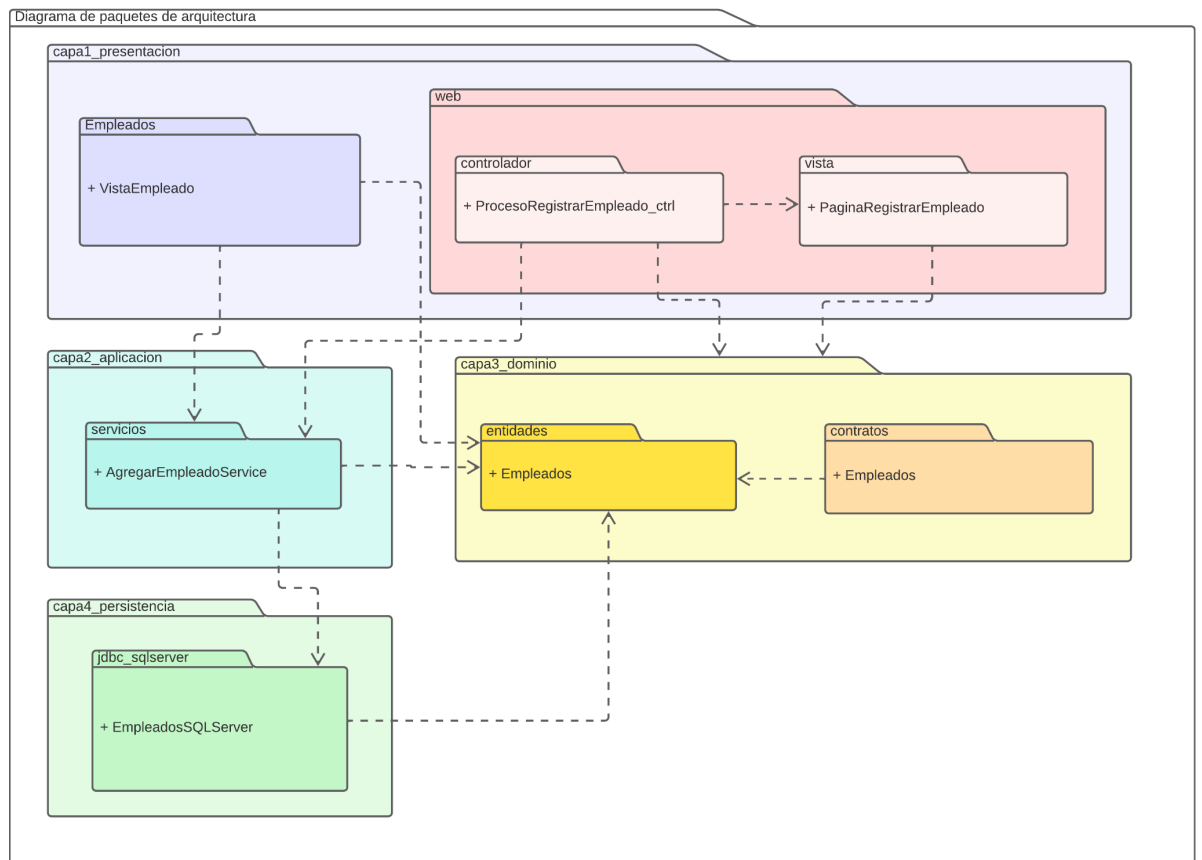


Figura 61. *diagrama de paquetes Registrar Empleado*

III. Conclusiones

En este primer incremento del proyecto, el sistema ha logrado un avance significativo al implementar la funcionalidad de guardar y gestionar los pedidos de los clientes de manera eficiente. Ahora, podemos mostrar en tiempo real qué mesas se encuentran disponibles para nuevos clientes y cuáles están ocupadas con pedidos pendientes o ya entregados. Este cumplimiento inicial de requisitos acordados con el restaurante Plaza Mayor es fundamental para asegurar que estamos en la dirección correcta.

No obstante, este es solo el inicio de nuestro proyecto. Nos comprometemos a continuar perfeccionando el sistema, ampliando sus capacidades y funcionalidades. Buscamos no solo satisfacer las necesidades básicas de un restaurante, sino también optimizar la experiencia del cliente. Nuestro objetivo es mejorar la atención al cliente y aumentar la eficiencia tanto en el servicio proporcionado por el personal como en la gestión de la cocina. Estamos emocionados por lo que está por venir y confiamos en que nuestro proyecto contribuirá de manera significativa al éxito y la eficacia operativa del restaurante Plaza Mayor.

IV. Referencias bibliográficas

TOVAR BERNARDO, R.K. 2019. "Sistema de Información para la Mejora de la Calidad de Servicio de Atención al Cliente en el Restaurant Campestre Los Girasoles". Tesis de pregrado. Universidad Nacional del Centro del Perú. [En línea]. Disponible en: https://repositorio.uncp.edu.pe/bitstream/handle/20.500.12894/5654/T010_76564165_T.pdf?sequence=1&isAllowed=y

FRANCY, P., 2022. Guías Prácticas [en línea]. Elaboración de diagramas de caso de uso , Universidad Cooperativa de Colombia Sede Villavicencio: Disponible en: <https://repository.ucc.edu.co/server/api/core/bitstreams/a7ea175f-de13-49fd-afba-c021baf7a6d0/content>

PIERRE, B. y RICHARD, F., 2014. *SWEBOK* [en línea]. Los Alamitos, CA, Estados Unidos de América: IEEE Computer Society Press. Disponible en: <https://ieeecs-media.computer.org/media/education/swebok/swebok-v3.pdf>.

García Hernández, María Dolores; Martínez Garrido, Cynthia A; Martín Martín, Naiara; Sánchez Gómez, Lorena. "La entrevista". Universidad de Centroamérica Jose Simeon Cañas. 2015, Disponible en : http://www2.uca.edu.sv/mcp/media/archivo/f53e86_entrevistapdfcopy.pdf

Davis, G. B., & Olson, M. H. (1984). Management information systems: conceptual foundations, structure, and development (2nd ed.). McGraw-Hill. <https://dl.acm.org/doi/10.5555/2519>

Guide to the software engineering body of knowledge. (s/f). Computer.org. Recuperado el 28 de septiembre de 2023, de <https://ieeecs-media.computer.org/media/education/swebok/swebok-v3.pdf>

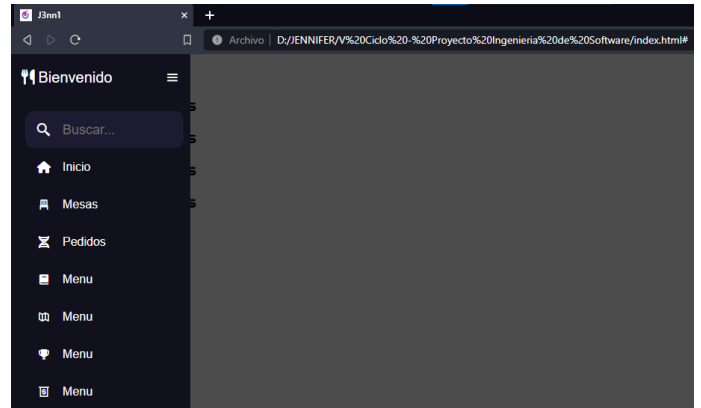
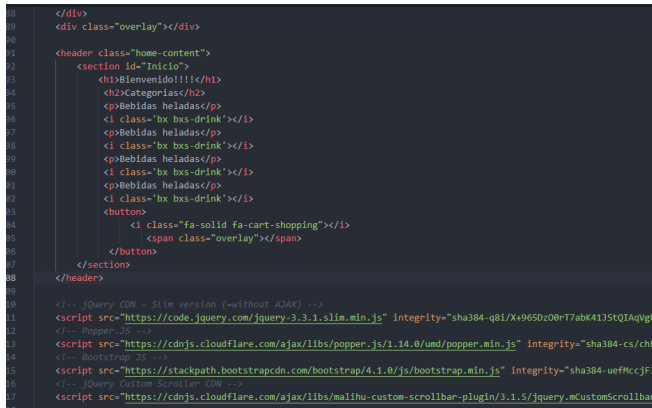
Yanirys Montes de Oca Hernández, Yuliesky Brito Díaz [en línea]. en el 2012, Módulo para la gestión de información de trámites protocolizables complejos en la notaría Buen Viaje de Santa Clara. Libros Eumed.net [en línea]. Disponible en: <https://www.eumed.net/libros-gratis/2012b/1232/index.html>

Wong Sandra. En el 2017, Análisis y requerimientos de software.
Disponble en :
https://repositorio.continental.edu.pe/bitstream/20.500.12394/4281/1/D_O_FI_N_103_MAI_UC0939_2018.pdf

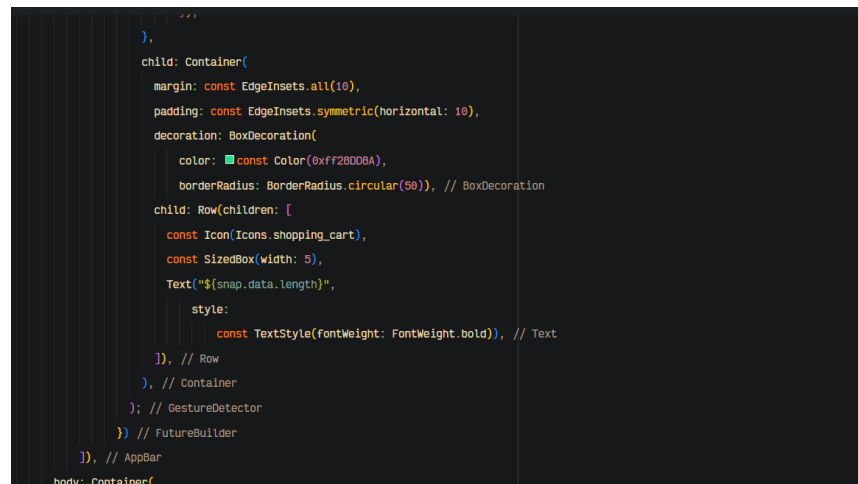
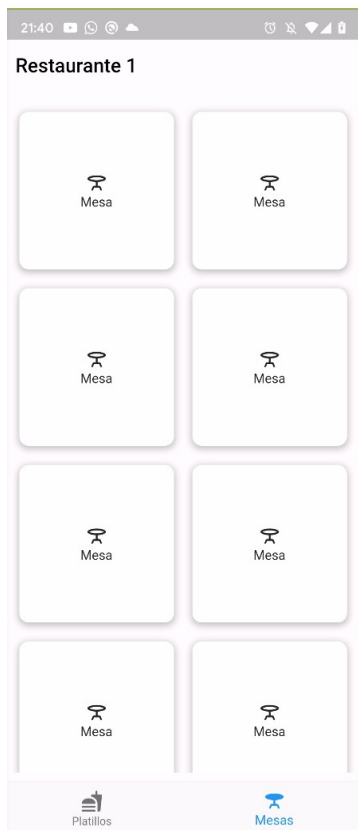
MANZANARES, Nilson Albeiro Ferreira, [b.r.]. Referencias bibliográficas | LENGUAJE DE MODELADO UNIFICADO UML [en línea]. Disponible en:
https://unadzsurlab.com/UML/U1/referencias_bibliogrificas.html

V. Anexos

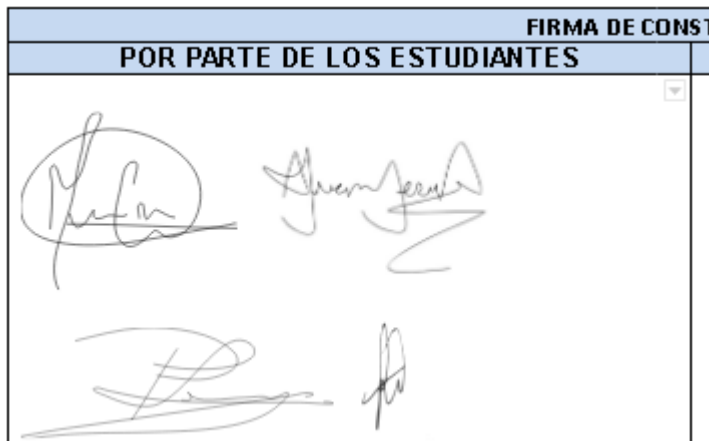
Página Web



App



PARTICIPANTES		
<p>Chileno Santisteban, Jennifer Paola(orcoid.org/0000-0001-9137-3352)</p> <p>Leon Yaipen, Piero Daniel (orcoid.org/0000-0003-1880-0084)</p> <p>Esquives Zapata, Juan Jesus (orcoid.org/0000-0002-6263-9745)</p> <p>Casas Castillo, Marcia Verónica (orcoid.org/0000-0002-7780-7844)</p>		
ORDEN DEL DÍA		
<ol style="list-style-type: none"> 1. Presentación de requerimientos 2. Presentación de prototipos 3. Diálogo sobre ideas 		
DESARROLLO DE LA REUNIÓN		
<p>Como se había mencionado, todo se había tratado a través de conversaciones con el representante, quien expresó su punto de vista sobre el proyecto y discutimos nuestras ideas en ese contexto. Estas interacciones fueron fundamentales para alinear nuestras propuestas con las expectativas del representante y garantizar un enfoque colaborativo en la realización del proyecto.</p>		
CIERRE		
<p>Durante las conversaciones que tuvimos sobre el trabajo, surgieron nuevas ideas y se realizaron modificaciones en el diseño, así como ajustes en algunos requerimientos que estaban correctos. Este proceso de discusión y análisis nos permitió enriquecer la propuesta inicial y garantizar que la planificación del proyecto esté más completa y precisa.</p>		
<p>Se ha distribuido equitativamente el trabajo propuesto, donde dos personas estarán a cargo del desarrollo del front end y otras dos se encargarán del back end, con el fin de lograr un mejor desarrollo del proyecto.</p>	Chileno Santisteban, Jennifer Paola	
	Leon Yaipen, Piero Daniel	
	Esquives Zapata, Juan Jesus	
	Casas Castillo, Marcia Verónica	



Link del repositorio

<https://github.com/Apo-Theddy/IngSoft-CartillaVirtual>

Link del video

<https://drive.google.com/drive/folders/1bQk2Rm-LvMQCM7Vz7YeRK4aOVB0tvVB7?usp=sharing>