**Requerimiento Formal del Sistema para la Plataforma de Gestión de Pedidos "Restaurante Catracho"**

**1. Introducción**

El objetivo del proyecto es desarrollar una plataforma para la gestión eficiente de pedidos, pagos, estados de pedidos, y lógica de colas dentro del Restaurante Catracho, con el fin de mejorar la experiencia del cliente, optimizar los procesos operativos y cumplir con los estándares de calidad de la ISO 9001:2015.

**2. Requerimientos Funcionales**

**2.2 Gestión de Pedidos**

* Crear, actualizar y eliminar pedidos.
* Asignación de prioridades a los pedidos basada en el tiempo de solicitud y el tiempo estimado de preparación.
* Envío de notificaciones automáticas al cliente sobre el estado de su pedido (en preparación, en ruta, entregado).

**2.3 Lógica de Colas**

* Organización de pedidos en función del tiempo de solicitud.
* Priorización automática de pedidos.
* Asignación de repartidores en función de su proximidad al restaurante.

**2.4 Seguimiento de Pedidos**

* Visualización en tiempo real del estado de los pedidos.
* Estimación del tiempo de entrega basado en APIs de geolocalización.

**2.5 Funciones Administrativas**

* Gestión de pedidos y su estado.
* Generación de reportes de ventas y métricas operativas.
* Acceso a funcionalidades de configuración del sistema para administradores.

**3. Requerimientos No Funcionales**

* **Rendimiento**: El sistema debe procesar pedidos y pagos en menos de 3 segundos.
* **Seguridad**: Implementar medidas de seguridad como cifrado de datos sensibles (especialmente pagos).
* **Escalabilidad**: La arquitectura debe permitir la expansión del sistema conforme crezca la demanda (uso de servicios en la nube como AWS o Azure).

**4. Estructura de Base de Datos**

La base de datos será relacional, utilizando MySQL o PostgreSQL. A continuación se presenta la estructura normalizada:

**4.1 Tablas Principales**

-- Creación de base de datos restaurantecatracho

CREATE DATABASE restaurante\_catracho\_db;

USE restaurante\_catracho\_db;

GO

CREATE TABLE Usuarios (

id\_usuario INT IDENTITY(1,1) PRIMARY KEY,

nombre NVARCHAR(255) NOT NULL,

correo NVARCHAR(255) UNIQUE NOT NULL,

contraseña NVARCHAR(MAX) NOT NULL,

rol NVARCHAR(50) CHECK (rol IN ('1', '2', '3', '4')) NOT NULL,

telefono NVARCHAR(20),

direccion NVARCHAR(MAX)

);

CREATE TABLE Productos (

id\_producto INT IDENTITY(1,1) PRIMARY KEY,

nombre NVARCHAR(255) NOT NULL,

descripcion NVARCHAR(MAX),

precio DECIMAL(10,2) NOT NULL,

categoria NVARCHAR(100),

disponible BIT DEFAULT 1

);

CREATE TABLE Pedidos (

id\_pedido INT IDENTITY(1,1) PRIMARY KEY,

id\_usuario INT NOT NULL,

estado NVARCHAR(50) CHECK (estado IN ('1', '2', '3', '4')) NOT NULL,

fecha\_creacion DATETIME DEFAULT GETDATE(),

fecha\_entrega\_estimada DATETIME,

monto\_total DECIMAL(10,2) NOT NULL,

FOREIGN KEY (id\_usuario) REFERENCES Usuarios(id\_usuario) ON DELETE CASCADE

);

CREATE TABLE Detalles\_Pedidos (

id\_detalle INT IDENTITY(1,1) PRIMARY KEY,

id\_pedido INT NOT NULL,

id\_producto INT NOT NULL,

cantidad INT CHECK (cantidad > 0) NOT NULL,

precio\_unitario DECIMAL(10,2) NOT NULL,

FOREIGN KEY (id\_pedido) REFERENCES Pedidos(id\_pedido) ON DELETE CASCADE,

FOREIGN KEY (id\_producto) REFERENCES Productos(id\_producto) ON DELETE CASCADE

);

**5. Arquitectura del Proyecto en Capas**

**5.1 Capa de Presentación (Frontend)**

* **HTML/CSS** para las interfaces de usuario.
* **JavaScript** (Vanilla) para la lógica dinámica:
  + Actualización en tiempo real de los estados de pedidos.
  + Validación de formularios de pedidos y pagos.
  + Integración con las APIs del backend.
  + Comunicación con el servidor usando Fetch API o XMLHttpRequest.

**5.2 Capa de Negocio (Backend)**

* **.NET Core o .NET 6**:
  + Controladores para la gestión de pedidos, pagos y seguimiento en tiempo real.
  + Lógica para el procesamiento de pagos (integración con Stripe o PayPal).
  + Implementación de la lógica de colas y la asignación de repartidores.
  + Servicios para la generación de reportes administrativos.

**5.3 Capa de Datos (Base de Datos)**

* **ORM**: Uso de **Entity Framework Core** para la interacción con la base de datos.
* Consultas optimizadas para la gestión de pedidos, pagos y reportes.

**6. Recomendaciones**

1. **Autenticación y Autorización**: Utilizar JWT (JSON Web Tokens) para la autenticación y autorización de usuarios (clientes, empleados, repartidores).
2. **Notificaciones**: Implementar un sistema de notificaciones push o por correo electrónico para informar a los usuarios sobre el estado de sus pedidos.
3. **Monitoreo de Desempeño**: Utilizar herramientas como **Azure Application Insights** o **AWS CloudWatch** para monitorear el rendimiento y la disponibilidad del sistema.
4. **Optimización de la Base de Datos**: Asegurarse de que las consultas estén bien indexadas para evitar cuellos de botella en el rendimiento, especialmente en las tablas de pedidos y pagos.
5. **Escalabilidad**: Considerar el uso de contenedores (Docker) y Kubernetes para facilitar la escalabilidad del sistema en caso de un alto volumen de usuarios.

**6. Recomendaciones**

**6.1. Relación Uno a Muchos (1:N)**

Ejemplo en tu caso:

* **Un usuario puede tener múltiples pedidos**, pero cada pedido pertenece a un solo usuario.
* **Un pedido puede tener múltiples detalles de pedido**, pero cada detalle pertenece a un solo pedido.

**Cómo se maneja en C# y EF Core:**

* Se define una propiedad de navegación en la clase principal (Usuario) que apunta a una lista de objetos (Pedidos).
* En la clase dependiente (Pedido), se coloca una propiedad de clave foránea (id\_usuario) y una propiedad de navegación (Usuario).
* EF Core infiere automáticamente la relación.

**6.2. Relación Muchos a Muchos (N:M)**

Ejemplo en tu caso:

* **Un pedido puede contener múltiples productos y un producto puede estar en múltiples pedidos**.
* Se necesita una **tabla intermedia** (Detalles\_Pedidos) que actúe como puente entre **Pedidos** y **Productos**.

**Cómo se maneja en C# y EF Core:**

* Se define una clase intermedia (DetallePedido) que tiene claves foráneas de Pedido y Producto.
* No se maneja la relación directamente entre Pedido y Producto, sino a través de la entidad DetallePedido.
* En DetallePedido, se incluyen referencias a Pedido y Producto, estableciendo las claves foráneas.

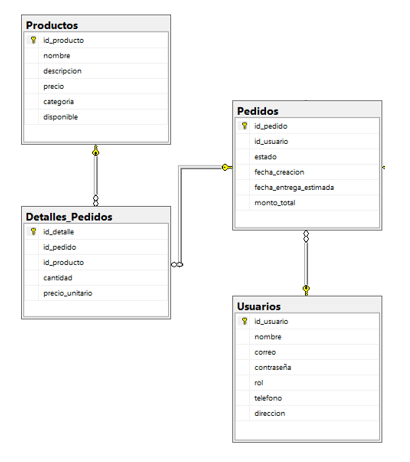
**6.3. Relación Uno a Uno (1:1)**

Ejemplo en tu caso:

* **Un pedido tiene un único pago** y **un pago pertenece a un único pedido**.

**Cómo se maneja en C# y EF Core:**

* Se establece una propiedad de navegación en ambos modelos (Pedido y Pago).
* Se define explícitamente la clave foránea en Pago apuntando a Pedido.
* EF Core lo configura automáticamente como una relación uno a uno.



Usuarios de prueba:

Para hacer pedidos cliente: pedidos@dondeelcatracho.com - 12345

Empleado: empleado@dondeelcatracho.com - 12345

Admin: admin@dondeelcatracho.com - 12345

Ambiente front end producción: https://r1h1.github.io/restaurantecatrachofrontend/

Ambiente back end producción: https://elcatrachorestaurantes.somee.com/swagger/index.html

Ambiente DB producción:

SQL Server version: MS SQL 2022 Express

SQL Server address: restaurante\_catracho\_db.mssql.somee.com

Login name: yeezuspm\_SQLLogin\_1

Login password: gev2vf7fn4

Connection string: workstation id=restaurante\_catracho\_db.mssql.somee.com;packet size=4096;user id=yeezuspm\_SQLLogin\_1;pwd=gev2vf7fn4;data source=restaurante\_catracho\_db.mssql.somee.com;persist security info=False;initial catalog=restaurante\_catracho\_db;TrustServerCertificate=True

Ambiente FTP producción:

Address: ftp://restaurante\_catracho\_db.backup.somee.com/restaurante\_catracho\_db\_MSSql\_Database\_Backup

Username: yeezuspm

Password: Yesus1821$

Acceso servidor producción:

https://somee.com/DOKA/Identity/Account/Login?ReturnUrl=%2FDOKA%2FDOU

User name or email: jesupalma9@gmail.com

Password: Yesus1821$

Repositorios:

Front End: https://github.com/r1h1/restaurantecatrachofrontend

Back End: https://github.com/r1h1/restaurantecatrachoapi