**Objetivo del Proyecto**

El objetivo principal de esta aplicación *Backend* es gestionar el sistema de reservas de almuerzos de un restaurante. El enfoque principal es desarrollar una API RESTful robusta que permita crear, consultar, modificar y cancelar reservas, aplicando una arquitectura por capas, validaciones estrictas y la persistencia de datos mediante Entity Framework Core (EF Core). Funcionalidades principales en orden de más a menos importante:

**CASOS DE USO, para la entidad TaskEntity (Mínimo 3):**

### **Prioridad Alta (CRUD básica y validaciones fundamentales)**

#### 1. Crear Reserva (InsertReserva)

* Descripción: Permite registrar una nueva reserva de almuerzo.
* Validaciones:
  + FechaReserva debe ser mayor o igual a la fecha actual y tener formato válido.
  + NumeroPersonas debe ser mayor a cero.
  + HorarioId debe existir en la tabla Horarios.
  + Se debe validar la disponibilidad (la reserva no debe exceder la CapacidadMaxima del Horario seleccionado, restando las reservas ya confirmadas/pendientes para esa fecha).
  + Si se proporciona ClienteId, debe existir el cliente.
  + Si no se proporciona ClienteId, los campos de NuevoCliente (Email, Nombre) no deben estar vacíos y el Email no debe existir previamente en la base de datos.
* Acciones:
  + Asignar automáticamente Estado = "Pendiente".
  + Si es un cliente nuevo, crearlo (AddCliente) y asignar el ClienteId generado a la reserva.
  + Retornar la reserva creada con su ReservaId generado y el código de estado 201 Created.

#### 2. Obtener todas las Reservas (GetAllReservas)

* Descripción: Devuelve la lista completa de reservas registradas.
* Opciones:
  + Filtros opcionales por FechaReserva (rango de fechas), Estado, ClienteId o HorarioId.
  + Paginación (pageNumber, pageSize).
* Validaciones:
  + Si se especifica ClienteId, debe existir el cliente.

#### 3. Obtener reserva específica (GetReservaById)

* Descripción: Devuelve una reserva según su ReservaId.
* Validaciones:
  + Si no existe la reserva, retornar excepción/código "Reserva no encontrada" (404 Not Found).
* Opcional:
  + Permitir incluir información del cliente, horario y detalles del pedido (DetallePedido).

#### 4. Modificar reserva (UpdateReserva)

* Descripción: Modifica campos de una reserva existente (ej. NumeroPersonas, HoraSolicitada dentro del mismo HorarioId o cambiando el HorarioId).
* Validaciones:
  + Verificar existencia de la reserva.
  + No permitir modificar el ReservaId o el ClienteId.
  + No se puede modificar una reserva con estado "Cancelada" o "Completada".
  + La modificación debe respetar la nueva disponibilidad (capacidad) del horario.

#### 5. Cancelar reserva (CancelarReserva)

* Descripción: Actualiza el estado de una reserva a "Cancelada" (eliminación lógica).
* Validaciones:
  + Verificar existencia de la reserva.
  + La reserva no debe tener un Pago asociado o, si lo tiene, se debe procesar un reembolso (lógica avanzada, opcional).
* Acciones:
  + Actualizar Estado = "Cancelada".
  + Retornar 204 No Content o 200 OK con un mensaje de éxito.

### **Prioridad Media (operaciones intermedias y mejoras)**

#### 6. Consultar Disponibilidad (CheckAvailability)

* Descripción: Determina qué horarios (Horario) tienen capacidad suficiente para una fecha y número de personas dado.
* Depende de: Horario y Reserva.
* Validaciones:
  + Fecha debe ser futura o actual.
  + NumeroPersonas debe ser mayor a 0.
* Acciones:
  + Calcular la suma de NumeroPersonas de reservas "Pendientes" y "Confirmadas" para cada Horario y Fecha.
  + Devolver solo los horarios donde CapacidadMaxima > (Reservas Ocupadas + NumeroPersonas solicitadas).

#### 7. Obtener Reservas por Cliente (GetReservasByCliente)

* Descripción: Devuelve todas las reservas asociadas a un ClienteId o Email específico.
* Validaciones:
  + Verificar existencia del cliente.

#### 8. Gestionar Pago de Reserva (ProcessPayment)

* Descripción: Registra un pago y, opcionalmente, cambia el estado de la reserva a "Confirmada".
* Depende de: Pago y Reserva.
* Validaciones:
  + La reserva debe existir y no estar en estado "Cancelada".
  + Verificar que la reserva no tenga un pago previo (restricción UNIQUE en Pago.ReservaId).
* Acciones:
  + Crear registro en Pagos.
  + Actualizar Reserva.Estado = "Confirmada".

### **Prioridad Baja (relaciones con otras entidades)**

#### 9. Registrar Detalle de Pedido (AddDetallePedido)

* Depende de: DetallePedido, Reserva, y Plato.
* Descripción: Permite asociar una pre-orden de platos a una reserva.
* Validaciones:
  + Verificar existencia de la reserva y del plato (PlatoId).
  + La Cantidad debe ser mayor a cero.
  + El plato debe estar activo.
* Acciones:
  + Registrar DetallePedido, usando el precio actual del Plato como PrecioUnitario.

## **REGLAS DE NEGOCIO APLICABLES A Reserva y Entidades** Relacionadas:

1. Integridad de la Reserva:
   * Una Reserva pertenece a un único Cliente y un único Horario.
   * No puede existir una reserva sin un ClienteId y HorarioId válidos.
2. Unicidad del Cliente:
   * El campo Email en la tabla Clientes debe ser único.
3. Estado Inicial:
   * El estado inicial de cualquier reserva siempre debe ser "Pendiente".
4. Validación Temporal:
   * FechaReserva y HoraSolicitada deben referirse a un momento en el futuro (o al menos a la fecha actual).
   * La hora solicitada debe estar dentro del rango (HoraInicio, HoraFin) del HorarioId seleccionado.
5. Restricciones de Cantidad (CHECK):
   * Reserva.NumeroPersonas debe ser mayor a 0.
   * DetallePedido.Cantidad debe ser mayor a 0.
6. Disponibilidad:
   * El sistema solo debe permitir la creación/modificación de reservas si la suma total de NumeroPersonas (de reservas activas) no excede la CapacidadMaxima del Horario para esa fecha específica.
7. Restricción de Pago:
   * Solo puede haber un pago por reserva (restricción UNIQUE en Pago.ReservaId).
8. Actualización Restringida:
   * No se puede modificar (PUT) ni registrar detalles de pedido (POST/Detalle) en reservas que ya han sido "Canceladas" o "Completadas".

**DIAGRAMA UML:**

