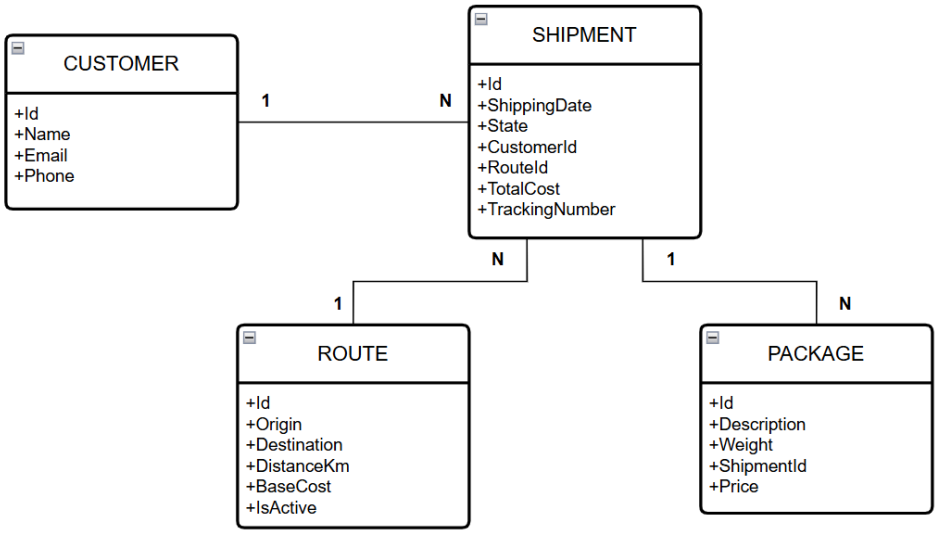
Proyecto API: Gestión de Envíos y Logística

Este proyecto tiene como objetivo desarrollar una API para la gestión de envíos y logística, permitiendo registrar clientes, crear envíos, asociar paquetes, definir rutas y rastrear el estado de cada envío. La solución está basada en una arquitectura RESTful implementada con ASP.NET Core, siguiendo el enfoque Code First con Entity Framework Core para la creación de la base de datos. La API utiliza DTOs, AutoMapper, validaciones con FluentValidation y un servicio central de validación (ValidatorService), siguiendo buenas prácticas de desarrollo modular y escalable.

1. **Casos de uso identificados.**

- **Registrar un cliente:** Permite crear nuevos clientes con su información básica (nombre, correo y teléfono).
- **Crear un envío (Shipment):** Registra un nuevo envío asociado a un cliente, con su respectiva ruta y estado inicial.
- **Asignar paquetes a un envío:** Permite agregar uno o más paquetes a un envío existente.
- **Actualizar el estado del envío:** Cambia el estado de un envío según su progreso logístico ("Pending", "In transit", "Delivered").
- **Consultar el estado/rastreo del envío:** Permite rastrear el envío por su número de seguimiento (Tracking Number).
- **Listar todos los envíos:** Muestra un resumen general con sus datos principales (cliente, ruta, costo total, estado).
- **Generar reporte de entregas**
  Permite obtener un resumen general de los envíos completados (Delivered) y sus métricas básicas.

2. **Entidad-relación de la base de datos.**



El modelo entidad-relación está compuesto por las siguientes entidades principales:

- Customer (Id, Name, Email, Phone).
- Shipment (Id, ShippingDate, State, CustomerId, RouteId, TotalCost, TrackingNumber).
- Package (Id, Description, Weight, ShipmentId, Price).
- Route (Id, Origin, Destination, DistanceKm, BaseCost, IsActive).

Relaciones:

- Un Customer puede tener varios Shipments.
- Un Shipment pertenece a un solo Customer y tiene múltiples Packages.
- Un Shipment está asociado a una sola Route.
- Una Route puede estar relacionada con varios Shipments.

3. Priorización de procesos.

En esta primera etapa del proyecto, se priorizó el módulo Shipment (Gestión de Envíos), ya que es el núcleo del sistema logístico y se relaciona directamente con las demás entidades (Customer, Route y Package).

- **Procesos priorizados:**
  1. CRUD completo de envíos (Shipment).
  2. Validaciones de negocio en la capa de servicio.
  3. Integración con AutoMapper y DTOs.
  4. Validaciones personalizadas con FluentValidation.
  5. Uso del ValidatorService centralizado para todos los modelos.
-
- **Validación de datos (ValidatorService):**
  - Validación del ID antes de consultas o actualizaciones.
  - Validaciones de negocio más específicas (estado del envío, fechas, etc.).

- **Integración parcial con otras entidades:**
  - Se implementaron referencias a Customer y Route, validando su existencia antes de crear nuevos envíos.

4. Casos de uso.

- **Caso de uso: Gestión de Envíos (Shipment CRUD)**

Este caso de uso permite al usuario crear, leer, actualizar y eliminar envíos registrados en la base de datos.
A través de los endpoints definidos en el ShipmentController, el sistema puede

interactuar con los envíos mediante operaciones básicas y validadas mediante el ValidatorService y reglas de negocio aplicadas en el ShipmentService.

- **Flujo principal:**

1. El usuario realiza una solicitud POST /api/shipment/dto o /dto/mapper.
2. El sistema valida:
    a. Existencia del cliente (CustomerId).
    b. Existencia y estado de la ruta (RouteId).
    c. Que la fecha de envío no sea futura.
    d. Que el número de seguimiento (TrackingNumber) sea único.
    e. Que el cliente no tenga más de 3 envíos simultáneos.
3. Si las validaciones son correctas, el envío se registra en la base de datos con estado Pending.
4. El usuario puede:
    a. Consultar (GET) los envíos existentes.
    b. Modificar (PUT) un envío.
    c. Eliminar (DELETE) un envío (solo si no está entregado).

- **Flujo alternativo:**
    - Si el cliente no existe → Error 400.
    - Si la ruta está inactiva → Error 400.
    - Si el tracking number ya existe → Error 400.
    - Si se exceden 3 envíos activos → Error 400.

- **Lógica aplicada en el código:**

    - Uso de AutoMapper para convertir entidades a DTOs y viceversa.
    - Uso de un servicio de validación (IValidatorService) para revisar datos antes de ejecutar operaciones.
    - Manejo de errores con BadRequest, NotFound e InternalServerError.
    - Retorno de respuestas tipadas con ApiResponse.

5. Criterios de aceptación y reglas de negocio.

Caso de uso: Gestión de Envíos

- **Criterios de aceptación:**

    - La API debe permitir crear, leer, actualizar y eliminar envíos correctamente.
    - Se deben validar CustomerId, RouteId, ShippingDate, TrackingNumber y State antes de insertar.

- El sistema debe devolver códigos HTTP adecuados (200, 201,204, 400, 404, 500).
- El número de seguimiento debe ser único y no vacío.
- El TotalCost debe ser mayor a 0.
- No se pueden registrar envíos en rutas inactivas.
- No se pueden eliminar envíos con estado “Delivered”.

- **Reglas de negocio implementadas:**

  - No se puede actualizar o eliminar un envío inexistente.
  - Las validaciones básicas de ID se realizan antes de acceder a la base de datos.
  - Las operaciones devuelven mensajes específicos si ocurren errores.
  - El ID del envío en la URL y en el cuerpo deben coincidir (PUT).
  - Cliente debe existir antes de crear el envío.
  - La ruta debe existir y estar activa.
  - El estado inicial de un envío será siempre “Pending”.
  - No permitir marcar 'Delivered' sin haber pasado por estados previos si aplica.
  - No se permitirá eliminar un envío con estado “Entregado”.
  - El número de seguimiento (TrackingNumber) será único.
  - TotalCost debe ser mayor a 0.
  - ShippingDate no puede ser anterior a hoy.
  - No permitir más de 3 envíos simultáneos por cliente.
  - Si una ruta está marcada como “inactiva”, no permitir registrar nuevos envíos en ella.