# APISERVICE: eCommerce Pole Products

## NEGOCIO:

APIService es una solución para flujo de datos pensada para un modelo de negocio adaptable y flexible. En este caso lo hemos querido configurar en base a la venta de productos de poledance, desde calzado y ropa adecuada hasta grips e incluso barras. Hoy en día aún es un mercado bastante reducido, pero con gran variedad de productos y precios (el precio podría depender de si el cliente es minorista o mayorista, socio o no de nuestro negocio, etc).

Mediante una interfaz amigable nos comunicamos con una base de datos a través de la API, con la que conseguimos centralizar todos nuestros datos (desde el tipo de usuario: si es administrador o comprador, hasta pedidos y productos) y poder manejarlos a nuestro gusto.

## OBJETIVOS PRINCIPALES:

1. Gestión del inventario de productos (en nuestro caso para la práctica de pole).
2. Gestión de base de datos tanto de usuarios (pudiendo ser un usuario un administrador o un comprador) como de productos y pedidos.
3. Gestión de la venta de productos con mayor eficacia y agilidad.

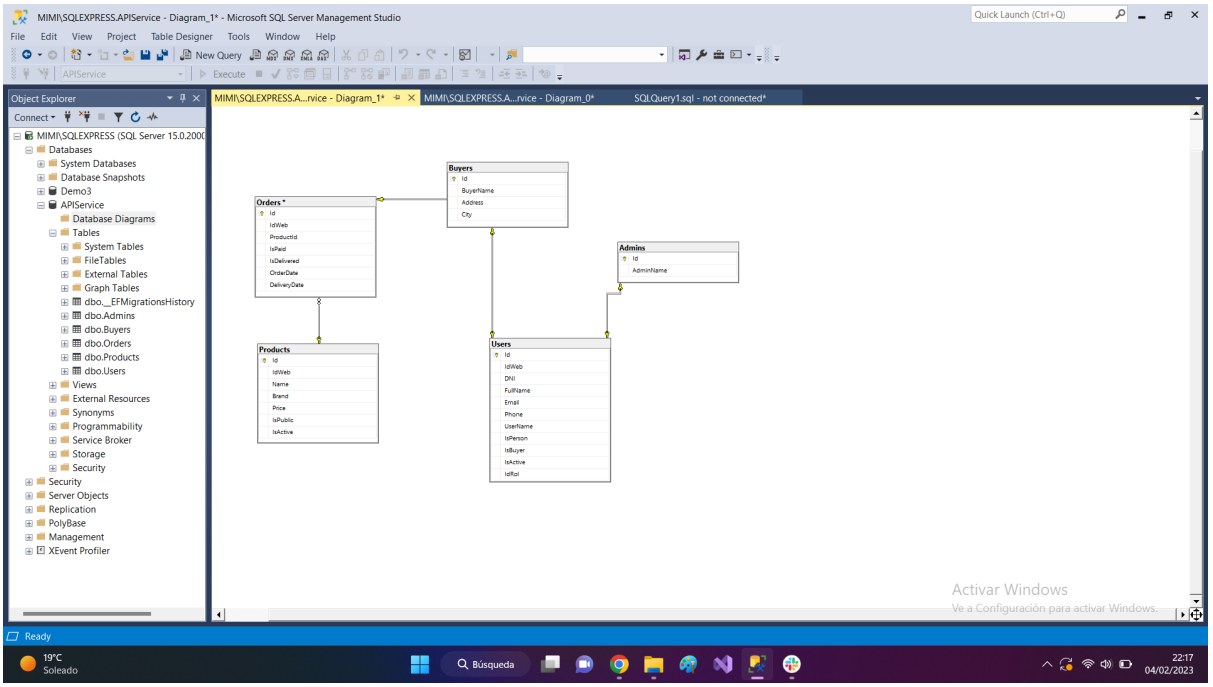
## ARQUITECTURA:

La arquitectura de la API tiene un diseño en capas independiente y limpio. Dentro de la solución está la capa principal APIService (que recoge los controllers y services, siendo ambos fundamentales) y, al mismo nivel, las capas Data (donde se encuentra el service context y las migraciones respecto de la base de datos), Entities y Logic. El número de capas va en función de la independencia, a mayor número de capas más seguridad de datos porque son más independientes entre ellos, ya que si queremos modificar datos habría que modificar también registros, la propia base de datos y, de esta forma, nos iríamos directamente a la capa que lo recoge.

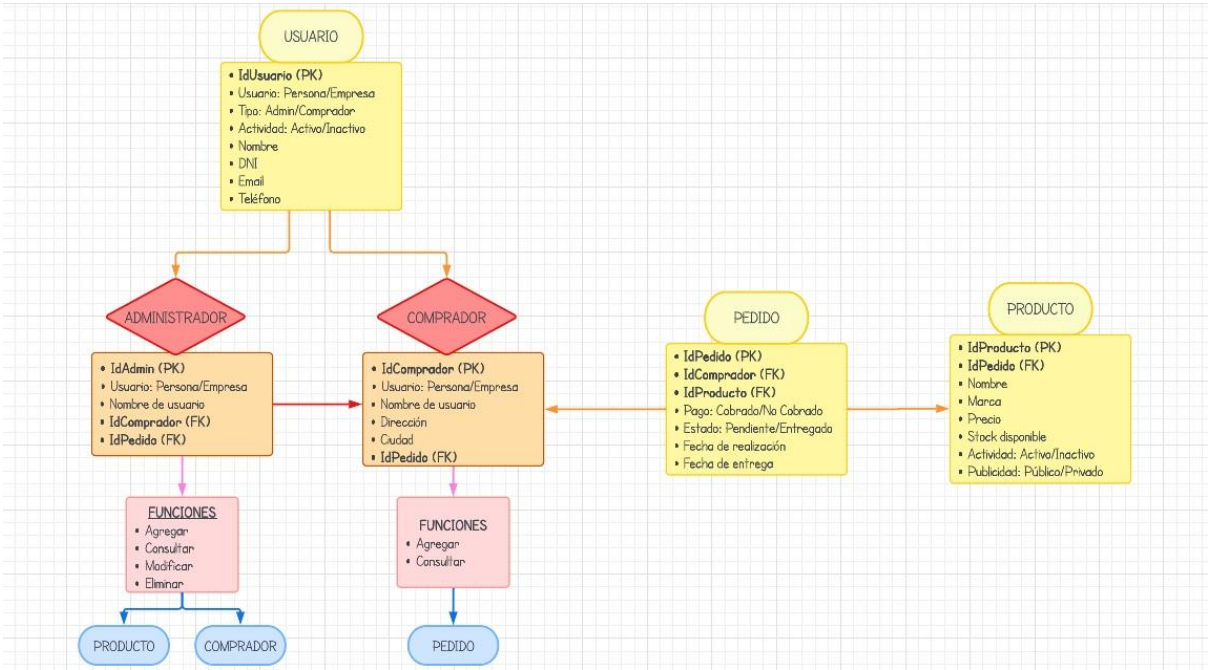
## SISTEMA RELACIONAL DE LA BASE DE DATOS:

Se han creado un total de cinco tablas, siendo User la clase padre con propiedades que recogen tanto datos básicos de una persona (nombre, DNI, teléfono, email..) como campos de activación y desactivación mediante un boolean (si el usuario es una persona o una empresa, si es administrador o comprador, si el perfil está o no activo). De esta clase User heredan dos clases hijos que serían Admin y Buyer (teniendo además atributos propios). Nuestra clase Admin tiene las cuatro funciones CRUD, tanto

para productos como para usuarios. Mientras que nuestra clase Buyer solo tiene acceso a agregar y consultar su pedido. Las dos tablas restantes serían Order y Product, cada una con sus propiedades correspondientes y haciendo uso de las Primary y Foreign Keys.




[1]Diagrama de la base de datos




[2] Diagrama realizado previamente al inicio del proyecto

[1] SQL Server Management Studio

[2] Lucidchart Diagrams

## TECNOLOGÍAS EMPLEADAS:

- FRAMEWORKS: Entity Framework v7.0 y .NET Core v6.0.
- PLATAFORMAS/SISTEMAS: Microsoft SQL Server Management Studio (base de datos en sí), Microsoft Visual Studio 2022 (donde construimos la API, que se comunica con la base de datos), GitHub (en repositorio en ramas), Lucidchart Diagrams (para el diseño previo de la estructura de nuestra base de datos), y Tablero Trello (para la organización de tareas y sus tiempos siguiendo metodología ágil SCRUM).
- LENGUAJES: C# y SQL.

## MÉTODOS IMPLEMENTADOS:

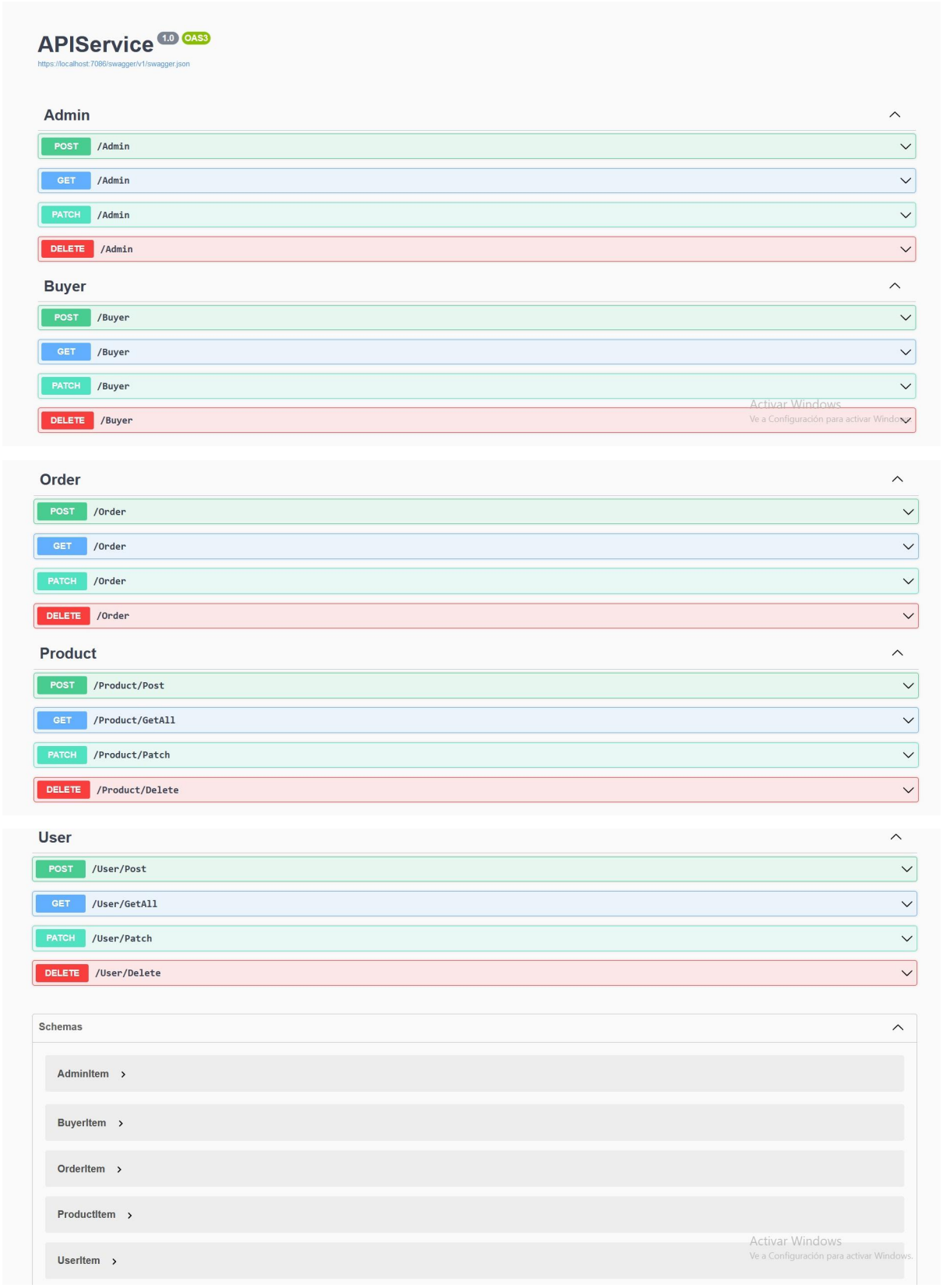
Los métodos implementados en nuestra API han sido CRUD (create, read, update y delete), esto es: crear, consultar, actualizar y eliminar los registros de nuestra base de datos. Como hemos mencionado, hemos hecho distinción entre el tipo de usuario, este puede ser administrador o comprador, siguiendo esto:

- El Administrador podrá insertar, modificar, eliminar y consultar tanto pedidos como compradores.
- El comprador podrá insertar y consultar productos, es decir, realizar pedidos y consultar su estado. En este modelo hemos querido que el comprador simplemente pueda añadir un producto por pedido. Pero más adelante podrá añadir más de un producto en un único pedido, es una de las mejoras previstas.

Estos métodos son:

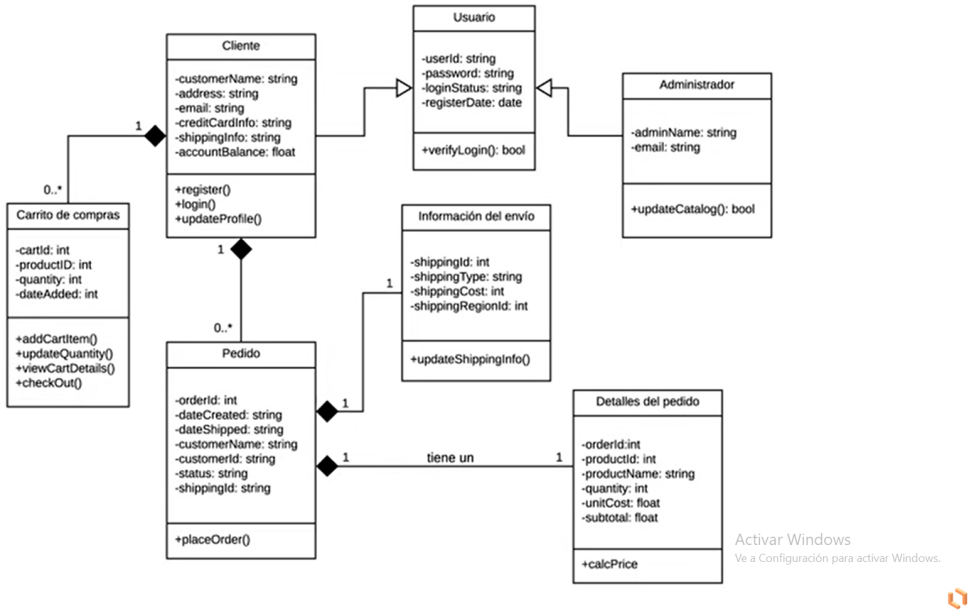
- POST: cuya funcionalidad es poder introducir nuestros registros (y, más adelante, con unos parámetros para que cualquiera no pueda introducir registros en nuestra base de datos, esto es la validación). Cuando ejecutemos el body se señalizarán los datos de esos registros que nos confirmará que se ha introducido adecuadamente nuestro registro.
- GET ALL: cuya funcionalidad es poder consultar todos nuestros registros y sus datos asociados (se quiere hacer distinción simplemente con el método get a través de un filtro, ya sea para filtrar productos por nombre, marca, etc). En el body se nos mostrará dicha consulta listada.
- PATCH: cuya funcionalidad es poder modificar registros en nuestra base de datos (igualmente, se añadirán los parámetros de validación). En el body estarán los datos del registro que queramos modificar.
- DELETE: cuya funcionalidad es poder eliminar registros de nuestras tablas (igualmente, más adelante, con unos parámetros para que un usuario no pueda borrar registros de nuestra base de datos sin permiso).

# DOCUMENTACIÓN FUNCIONAL DE LA WEB API:



## EXTENSIONES Y MEJORAS DE EXPERIENCIA DE USUARIO:

- Introducir validaciones: usuario y contraseña que serán introducidos por el propio usuario, que tendrán un rol de usuario asociado automáticamente por nuestro programa. A su vez, en los métodos como hemos indicado anteriormente.
- Crear un front accesible para el cliente con un login de usuario, distinguiendo entre administrador y comprador, y dependiendo de la selección le lleve a una interfaz con unas u otras funciones.
- El comprador tendrá la posibilidad de añadir más de un producto en un solo pedido.
- El comprador podrá ser socio o no de nuestra plataforma, a través de un boolean.
- Cálculos para pedidos, por ejemplo el precio final del pedido con más de un producto, o aplicar descuento si el comprador es socio o no.
- Gestión del stock con reducción del stock cada vez que se realice un pedido.
- Notificación automática mediante email, por ejemplo cada vez que un usuario realice una compra se le notifique por correo electrónico, o del estado de su pedido (ya ha sido enviado o entregado).
- Estaría bien implementar, lo que conllevaría un cambio del modelo de negocio, que un comprador pueda también vender sus propios productos, al estilo Vinted o Wallapop, añadiendo a su vez secciones productos nuevos y de segunda mano.




[3] Diagrama mejorado (relaciones de herencia y composición, con multiplicidad)

[3] Lucidchart Diagrams

**ENLACES:**

- Repositorio Git: https://github.com/miriamfactoriaf5/APIService
- Lucidchart Diagrams: https://lucid.app/lucidchart/89edc91c-fe8e-42b8-8301-f2c1207c7cc4/edit?invitationId=inv_cf39427b-eb1a-4076-9a8b-35c01776fed6&page=0_0#
- Tablero Trello: https://trello.com/b/Nb0UqLlj/team2