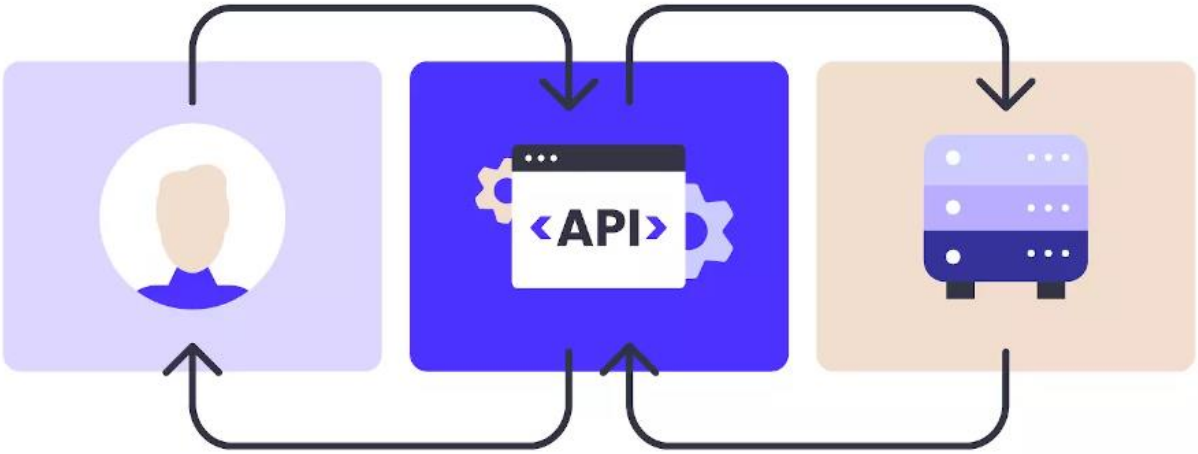


Backend Challenge Orenes

DESIDERIO ALMANSA PORRERO

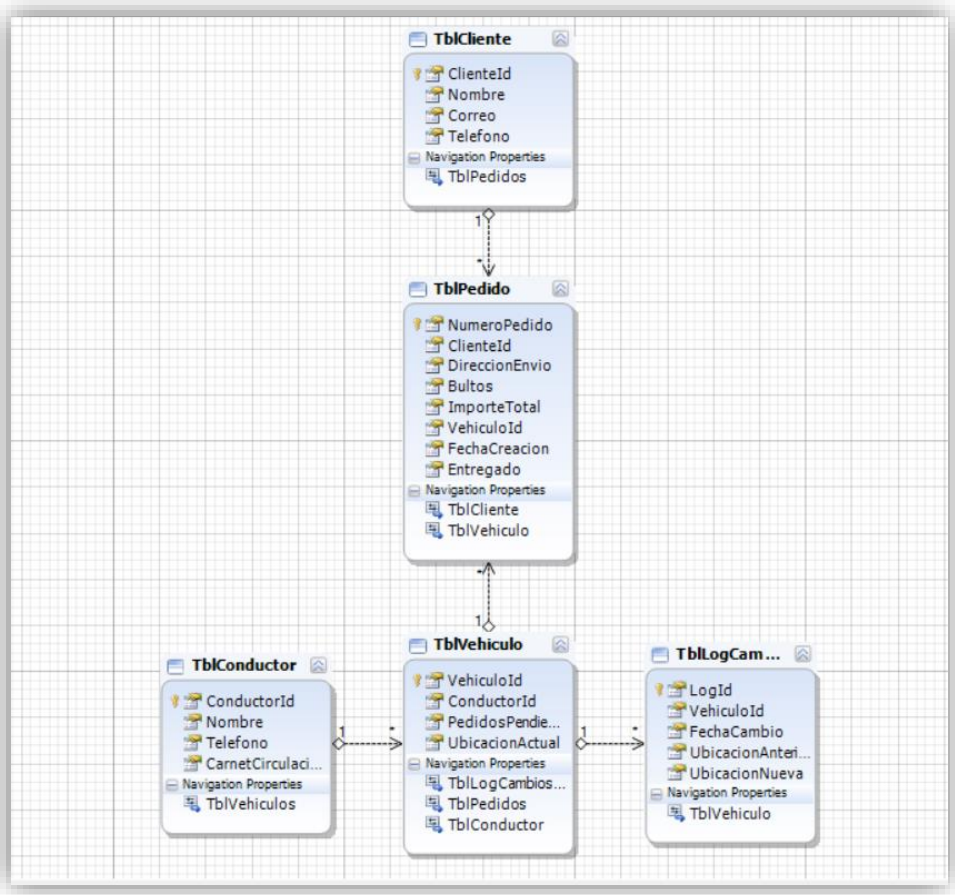
23/09/2022

INDICE

1. Tecnologías empleadas.

- **Entorno de desarrollo**: Visual Studio 2019
- **Framework**: .NET Core 3.1
- **Lenguaje de programación**: C#
- **Base de datos**: SQLServer
- **Pruebas para la API**: Postman

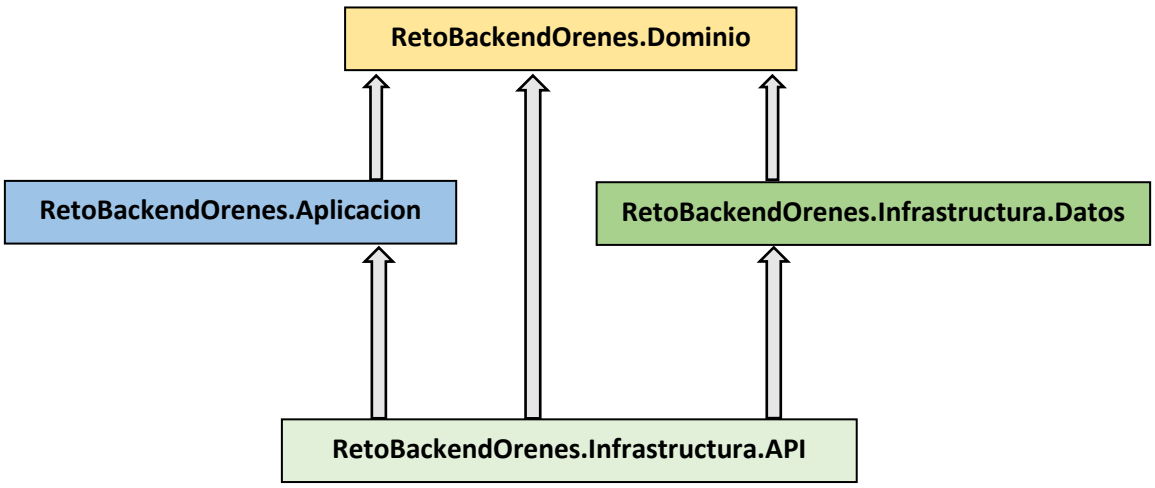
2. Modelo de datos.



(Se adjunta junto con el proyecto un script para crear las tablas en la base de datos y otro para rellenarlas con algunos registros, por si fuesen necesarios).

3. Arquitectura.

Se ha seguido una **Arquitectura Hexagonal**, lo que nos permite una gran escalabilidad y mantenimiento del proyecto. A continuación, se muestra un diagrama de dependencias.



- **RetoBackendOrenes.Dominio:** En este proyecto se encuentran las entidades de dominio y las interfaces. Habrá una interfaz para cada una de las funciones básicas CRUD (Agregar, Editar, Eliminar y Listar). Estas interfaces se encuentran en la carpeta *Interfaces*. Dentro de esta misma carpeta encontramos otra llamada *Repositorios*, en la que se encuentra la interfaz de repositorio base y otra interfaz para las transacciones.

- **RetoBackendOrenes.Aplicación:** En este proyecto se encuentran los casos de uso, proporcionados por interfaces. En el podemos encontrar una carpeta llamada *Interfaces*, que contiene la interfaz de servicio base, que hereda de las interfaces CRUD creadas en la capa de dominio, y otra carpeta llamada *Servicios*, que contiene los casos de uso correspondientes para cada entidad. Cada uno de ellos hereda de la interfaz de servicio base mencionada.

- **RetoBackendOrenes.Infrastrcutura.Datos:** En este proyecto se encuentra el contexto y las configuraciones necesarias para poder entablar conexión con la base de datos. Además en la carpeta *Repositorios* encontramos una clase para cada repositorio de cada entidad. En cada uno de ellos se implementa la lógica que ha de seguir las funciones CRUD anteriormente establecidas en la interfaz de repositorio base.

  Además, contamos con una clase llamada *Program.cs* que debemos ejecutar en caso de que la base de datos no esté creada.

- **RetoBackendOrenes.Infrastrcutura.API:** Se trata del proyecto de tipo API REST. Contiene un controlador para cada entidad de la capa de dominio, con sus respectivas consultas (GET, POST, PUT y DELETE). Adicionalmente, se le ha agregado **Swagger** para poder documentarla.

- **RetoBackendOrenes.WinformsPruebas:** Este proyecto se crea con el fin de probar las principales consultas que se piden en el enunciado del reto: asignar pedidos a un vehículo, actualizar ubicación del vehículo y obtener la ubicación y el vehículo del pedido.

4. API REST.

- **ClienteController:** Controlador para la entidad Cliente. Accedemos a él a través del endpoint *api/cliente/*<llamada>. Contiene las siguientes llamadas:
    - GET api/cliente/getall: devuelve una lista con todos los clientes
    - GET api/cliente/{id}: devuelve el cliente al que pertenezca el id indicado.
    - POST api/cliente: inserta el cliente, que se le pase a través del body, en la base de datos.
    - PUT api/cliente/{id}: edita el cliente indicado por el id que se le pasa, con la información que almacena el body de la petición.
    - DELETE api/cliente/{id}: elimina el cliente al que pertenezca el id indicado.

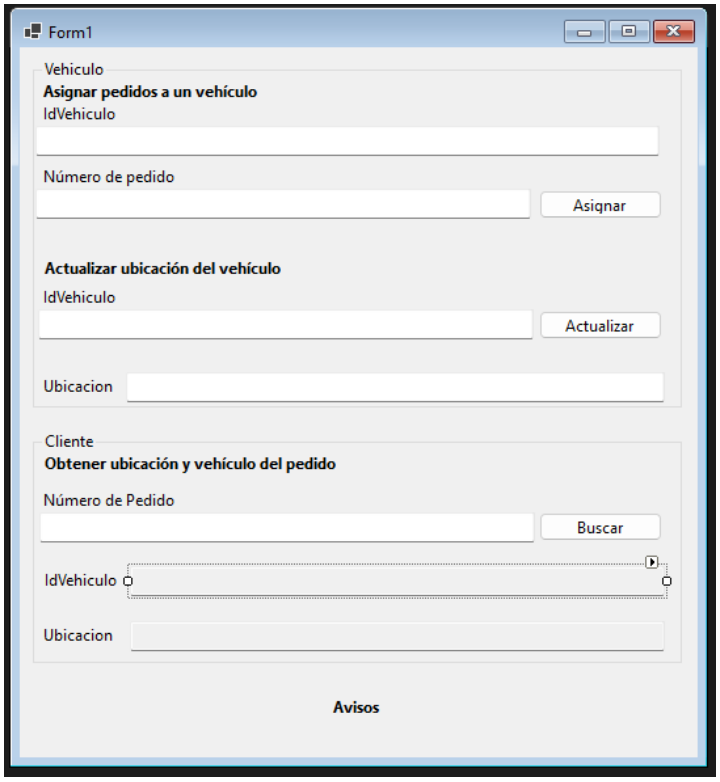
- **ConductorController:** Controlador para la entidad Conductor. Accedemos a él a través del endpoint *api/conductor/*<llamada>. Contiene las siguientes llamadas:
    - GET api/conductor/getall: devuelve una lista con todos los conductores
    - GET api/conductor/{id}: devuelve el conductor al que pertenezca el id indicado.
    - POST api/conductor: inserta el conductor, que se le pase a través del body, en la base de datos.
    - PUT api/conductor/{id}: edita el conductor indicado por el id que se le pasa, con la información que almacena el body de la petición.
    - DELETE api/conductor/{id}: elimina el conductor al que pertenezca el id indicado.

- **LogCambiosUbicacionesController:** Controlador para la entidad LogCambiosUbicacion. Accedemos a él a través del endpoint *api/logcambiosubicacion/*<llamada>. Contiene las siguientes llamadas:
    - GET api/logcambiosubicacion/getall: devuelve una lista con todos los logs
    - GET api/logcambiosubicacion/{id}: devuelve el log al que pertenezca el id indicado.
    - POST api/logcambiosubicacion: inserta el log, que se le pase a través del body, en la base de datos.
    - PUT api/logcambiosubicacion/{id}: edita el log indicado por el id que se le pasa, con la información que almacena el body de la petición. (comentado ya que no se debe editar un log)
    - DELETE api/cliente/{id}: elimina el log al que pertenezca el id indicado.

- **PedidoController:** Controlador para la entidad Pedido. Accedemos a él a través del endpoint *api/pedido/*<llamada>. Contiene las siguientes llamadas:
    - GET api/pedido/getall: devuelve una lista con todos los pedidos.
    - GET api/pedido/{numPedido}: devuelve el pedido al que pertenezca el numPedido indicado.
    - GET api/pedido/obtenerVehiculo/{numPedido}: devuelve el vehículo que tenga asignado el pedido con el numPedido que se le pasa.
    - POST api/pedido: inserta el pedido, que se le pase a través del body, en la base de datos.
    - PUT api/pedido/{numPedido}: edita el pedido indicado por el numPedido que se le pasa, con la información que almacena el body de la petición.
    - PUT api/pedido/asignarVehiculo/{numPedido}/{idVehiculo}: asigna al pedido indicado con numPedido, el vehículo indicado con idVehículo.
    - DELETE api/pedido/{id}: elimina el pedido al que pertenezca el id indicado.

- **VehiculoController:** Controlador para la entidad Vehiculo. Accedemos a él a través del endpoint *api/vehiculo/*<llamada>. Contiene las siguientes llamadas:
    - GET api/vehiculo/getall: devuelve una lista con todos los vehículos.
    - GET api/vehiculo/{id}: devuelve el vehiculo al que pertenezca el id indicado.
    - POST api/vehiculo: inserta el vehiculo, que se le pase a través del body, en la base de datos.
    - PUT api/vehiculo/{id}: edita el vehiculo indicado por el id que se le pasa, con la información que almacena el body de la petición.
    - PUT api/vehiculo/cambiaubicacion/{id}/{nuevaUbicacion}: cambia la ubicación que pasamos a través del parámetro nuevaUbicación, del vehículo indicado con id.
    - DELETE api/vehiculo/{id}: elimina el vehiculo al que pertenezca el id indicado.

5. Winforms de prueba.



Para realizar las peticiones principales que se solicitan en el enunciado, se ha creado este sencillo winforms. En los textboxs podemos introducir los datos solicitados y tras pulsar el botón correspondiente a cada petición, nos realizará la llamada a la API. Tanto si se realiza la petición correctamente como si no, abajo del todo se mostrará un mensaje indicándonoslo (label *Avisos*).

- **Asignar pedidos a un vehículo:** realiza la petición PUT api/pedido/asignarVehiculo/{numPedido}/{idVehiculo}. Se actualiza el cambo vehiculoId del pedido.
- **Actualizar ubicación del vehículo:** Realiza la petición PUT api/vehiculo/cambiaubicacion/{id}/{nuevaUbicacion}. Se actualiza la ubicación de un vehículo y se genera un registro en la tabla LogCambiosUbicación.
- **Obtener ubicación y vehículo del pedido:** realiza la petición GET api/pedido/obtenerVehiculo/{numPedido}.

6. Ejecución.

Para poder probar el sistema correctamente debemos de seguir los siguientes pasos:

1. Teniendo SQLServer instalado, ejecutar el *Program.cs* que se encuentra en el proyecto RetoBackendOrenes.Infrastructura.API. Esto nos generará la base de datos con las tablas de cada entidad de la capa de Dominio, en el caso de que no esté creada aún.
2. Debemos ejecutar el script sql *cargarDatos* (que se encuentra en la carpeta *Documentacion y scripts sql* del repositorio)para rellenar la tablas con datos de ejemplo (en caso de que no se quiera insertar manualmente).
3. Ejecutar la solución. Esto hará que se arranque la API al estar configurado este proyecto como proyecto de inicio.
4. Finalmente iniciar una instancia del proyecto RetoBackendOrenes.WinformsPrueba, lo que hará que se ejecute el winforms mencionado anteriormente. Para ello debemos pulsar click derecho sobre este proyecto > Depurar > Iniciar nueva instancia.