He utilizado .Net Core v6, ya que es la versión estable más moderna y contiene Swagger para poder probar la API.

Antes de ejecutar el programa por favor, revisa que están instalados los paquetes Nugget "Microsoft.EntityFrameworkCore.SqlServer", "Microsoft.EntityFrameworkCore.Tools" y "Microsoft.EntityFrameworkCore.Design" en la versión 6.0.14, esto permite que se puedan utilizar las herramientas para crear y manejar comandos en la base de datos.

Y ejecuta los siguientes comandos poder realizar peticiones a la API:

- `add-migration initial`
- `update-database`

Para crear la base de datos he utilizado el paquete Nugget "Microsoft.EntityFrameworkCore.SqlServer" porque crea una base de datos en el propio Visual Studio sin necesidad de instalar programas adicionales. Elegí esta opción debido a las limitaciones del ordenador que he utilizado que no me permitía instalar otros programas, aunque una mejor opción hubiera sido crear la BBDD en Sql Server Management Studio y posteriormente generar un script para que al probarlo fuera más limpio y tenerlo separado del código.

Según lo que he entendido del reto, me he centrado en crear una API funcional lo más sencilla posible que garantice que se cumplen con los requisitos indicados.

He creado 3 modelos: Car, Client y uno que utiliza información de los dos anteriores Ren para los alquileres propiamente.

Car contiene el inventario de coches y con el método Get en el controlador podemos ver la información de los coches y si se encuentran o no alquilados.

Client contiene la información del cliente y sus puntos.

Rent por sencillez los días de reserva se indican como un entero y luego se añaden los días extra, para no realizar cálculos sobre fechas, aunque en un entorno totalmente realista se deberían manejar fechas concretas.

Para tener más ordenado en código y poder reutilizar en caso de ser necesario, los cálculos de los precios y de los puntos de fidelidad se realizan en 2 clases distintas dentro de la carpeta Business, “RentalService” y “” respectivamente.

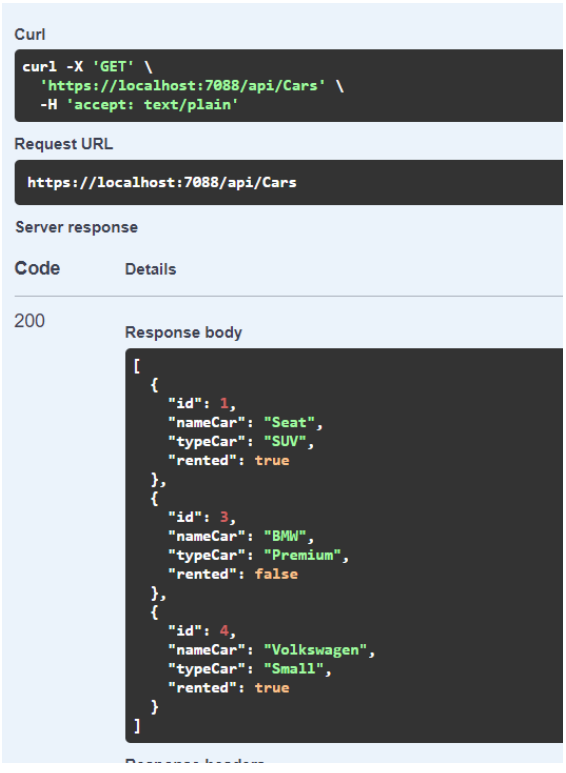
Para las funciones pedidas:

- Inventario de coches: se pueden ver los coches con el método Get del controlador CarController. Indica el id, el coche, el tipo (se controlan que solo puedan ser los tipos 3 indicados) y si esta alquilado o no.
- Calcular el precio del alquiler: en el método Post del controlador RentController se realiza una llamada al método “CalculatePrice” de la clase auxiliar “RentalService”, donde se calculan los precios de los alquileres para cada tipo de vehículo.
- Mantener la traza de los puntos de lealtad: en el método Post del controlador RentController se realiza una llamada al método “CalculatePoints” de la clase auxiliar “LoyaltyPoints”, donde se calculan los puntos de los clientes según el tipo de vehículo en el momento de alquilar. Se acumulan a los puntos que ya tuvieran los clientes.

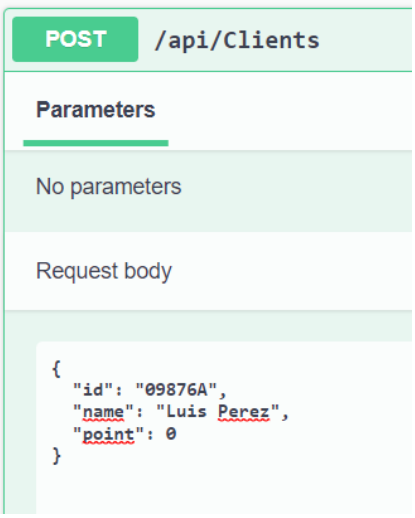
A continuación, se muestran imágenes de algunas pruebas:

- Creación de un Coche

- Inventario de coches



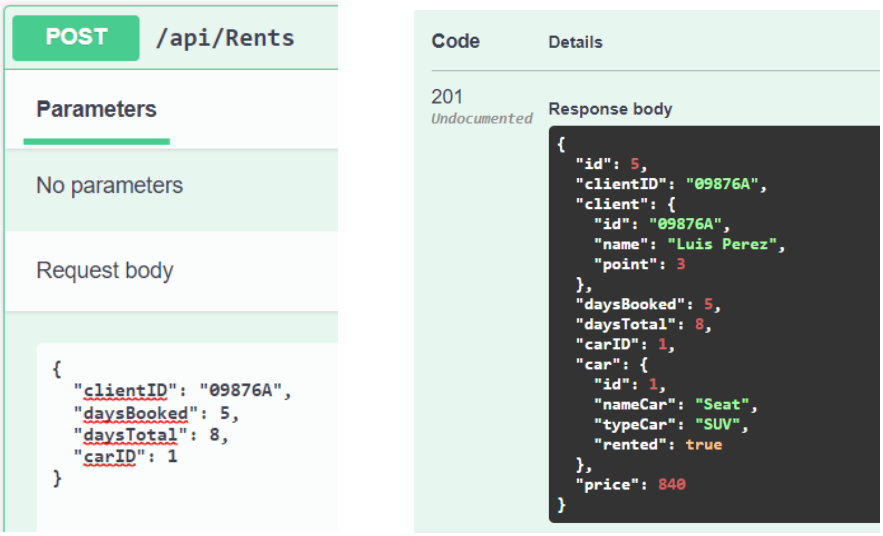
- Creación de un cliente



- Listado de Clientes con sus puntos

- Creación de un alquiler dentro del plazo acordado

- Creación de un alquiler con días extra



- Mostrar clientes con sus puntos actualizados tras realizar alquileres

Curl

```
curl -X 'GET' \
  'https://localhost:7088/api/Clients' \
  -H 'accept: text/plain'
```

Request URL

```
https://localhost:7088/api/Clients
```

Server response

Code	Details
200	Response body

```
[
  {
    "id": "09876A",
    "name": "Luis Perez",
    "point": 3
  },
  {
    "id": "12345",
    "name": "Cris",
    "point": 7
  }
]
```