**Nombre: José Ricardo Martinez Ruano**

Prueba técnica

# Arquitectura de la solución

La solución ha sido desarrollada en .NET y consta de dos proyectos principales:

* API (estadoCuentaAPI): la cual es el Backend en .NetCore que expone los endpoints para manejar datos y lógica del negocio.
* MVC con Razor (estadoCuentaUI): es la aplicación Web que consume la API y proporciona la Interfaz de Usuario para la administración y visualización de los datos.

## Estructura de proyectos:

* API (Backend):
  + Controllers: Manejan las solicitudes HTTP y actúan como intermediarios entre el cliente y la lógica de negocio.
  + DTOs: se usan para transferir datos entre el backend y el frontend, evitando exponer entidades de la base de datos directamente.
  + Interfaces: Definen contratos para los servicios, promoviendo la inyección de dependencias.
  + Mappings: transforma los datos entre entidades a DTO, facilitando la separación de responsabilidades
  + Model: representa las tablas de la base de datos y su relación con la capa de datos.
  + Services: contiene la lógica de negocio y la implementación de las interfaces
  + Validators: validaciones que se realizan a los datos antes de poder ser enviados a la base o a la vista
* MVC (Frontend con Razor)
  + Vistas: Plantillas HTML con Razor para mostrar información dinámica.
  + Controllers: Manejan peticiones desde el frontend y gestionan la navegación entre vistas.
  + Wwwroot: con las carpetas de css y js personalizados para agregar los estilos y las funcionalidades de los botones

## Comunicación entre proyectos

El proyecto MVC consume la API mediante **HttpClient**, enviando y recibiendo datos en formato JSON. Se utilizan los métodos HTTP:

* Métodos GET, POST, PUT y DELETE para interactuar con la API.
* En esta prueba los métodos PUT (para actualizar) y el DELETE (para borrar) no se han utilizado, pero se podrían usar para actualizar información de la tarjeta de crédito como por ejemplo cambiar la fecha de corte, cambiar el limite de la tarjeta, etc. Para el caso de borrar se podría, por ejemplo, borrar un cliente si este no tiene tarjeta asignada

## Seguridad y manejo de datos

* Los DTOs son usados para evitar exponer modelos internos como las tablas de la base de datos o bien para crear un modelo personalizado de dato donde no se utiliza toda la información como por ejemplo para los pagos se usa un DTO llamado PagoDTO.
* Con la configuración de CORS en la API es usada para permitir peticiones desde el frontend al backend, en esta configuración se le dice por ejemplo que el sitio (o IP) es de confianza y que puede dejar pasar las peticiones, esta dirección esta almacenada en el appsettings
* Procedimientos Almacenados: Utilizados para optimizar consultas en SQL Server, asegurando mayor eficiencia en la obtención de datos

## Librerías y Herramientas Implementadas

Para mejorar la estructura y funcionalidad de la aplicación, se han considerado e implementado las siguientes librerías:

1. **FluentValidation (Validación de Datos):** Permite definir reglas de validación en clases separadas, en lugar de validar directamente en los controladores o modelos.

* Se ha intentado implementar **FluentValidation** en la API para validar las entradas de datos de forma más estructurada y reutilizable.

1. **AutoMapper (Mapeo de Objetos):** ayuda a mantener la separación entre la lógica de negocio y las estructuras de datos que se exponen a los clientes.

* Se ha buscado integrar **AutoMapper** para transformar entidades del dominio en **DTOs (Data Transfer Objects)** y viceversa.

1. **Generación y Exportación de Archivos:** Se han explorado librerías para la exportación de archivos en diferentes formatos:

* **PDF:** Se agrego la librería **iText7** para generar reportes en formato PDF desde la API en este caso para la exportación del Estado de cuenta.
* **Excel:** Se ha utilizado **ClosedXML** para la generación de archivos Excel, facilitando la exportación de datos tabulares como en el caso de exportar los movimientos de las compras realizadas por la tarjeta de crédito y en un rango de fechas.

1. **HealthCheck**

* Es útil para verificar si la aplicación está funcionando correctamente y si todos los servicios necesarios (como bases de datos, servicios externos, etc.) están operativos.

|  |  |  |
| --- | --- | --- |
| **Tecnología/Librería** | **Propósito** | **Estado de Implementación** |
| **Entity Framework Core** | ORM para la gestión de la base de datos. | Implementado |
| **AutoMapper** | Mapeo entre entidades y DTOs. | En uso |
| **FluentValidation** | Validación de datos de entrada. | Parcialmente implementado |
| **iText7** | Generación de archivos PDF. | Implementado |
| **ClosedXML** | Exportación de datos a Excel. | Implementado |
| **Swashbuckle (Swagger)** | Documentación de la API. | Implementado |
| HealthCheck | monitorear la salud de una aplicación. | Implementado |

# Endpoints

Los Endpoints utilizados son:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Controller | Endpoint | Descripción | Parámetros | Retorno |
| Cliente | crear | crear un nuevo cliente en la base de datos | ClienteDTO: Nombre: nombre del cliente, apellido: apellidos del cliente, dui: documento de identificación | Códigos  201: cliente creado exitoso  400 si hubo algún error de validación 500: si hubo algún fallo en la base de datos |
| Cliente | [HttpGet("{clienteId}")] | Obtener la información de un cliente por su id | clienteId: Identificador del cliente en la tabla | Códigos:  200 con la información del cliente  404 si no se encuentra el cliente  500 si hubo un error |
| EstadoCuenta | ObtenerEstadoCuenta | Obtener información del estado de cuenta de una tarjeta de crédito especifica | tarjetaId: Id de la tarjeta de crédito | códigos:  200 con la información del estado de cuenta  404 si no se encuentra el estado de cuenta  500 si hubo un error |
| Export | ExportarEstadoCuentaPDF | exporta el estado de cuenta en un archivo PDF | tarjetaCreditoId: Identificador de la tabla Tarjeta de crédito asociado al cliente | El archivo en PDF del estado de cuenta y los códigos:  200 si se generó correctamente  500 si hubo un error |
| Export | ExportarComprasExcel | Excel de los movimientos de las compras realizadas por el cliente en un rango de fechas especificado | fechaFin: fecha de fin de la consulta  fechaInicio: Fecha de inicio de la consulta  tarjetaCreditoId: Id de tarjeta de crédito a consultar | Un archivo en Excel o el contenido del mensaje de error  200 si se generó correctamente  400 si los parámetros no son válidos  404 si no se encontraron compras en el rango de fechas especificado  500 si hubo un error |
| HistorialTransacciones | ObtenerHistorialTransacciones | obtener historial de transacciones de la tarjeta de crédito especificada por ID | tarjetaCreditoId: ID de la tarjeta de credito | Códigos: 200 con el historial de transacciones o un error  500 si ocurre algún fallo |
| Movimiento | RegistrarCompra | Registro de compras de tarjeta de crédito | MovimientoCompraDTO: TarjetaCreditoId: ID de la tabla tarjeta de credito  FechaMovimiento: fecha en la que se realizó la compra  Descripcion: de la compra registrada  Monto: de la compra | Códigos: 200 cuando es exitoso  400 si la tarjeta no se encuentra  500 si hubo un error |
| Movimiento | RegistrarPago | Registro de pago de tarjeta de crédito | PagoDTO (objeto con las propiedades para crear un movimiento de pago):  TarjetaCreditoId: ID de la tabla tarjeta de crédito  FechaMovimiento: fecha en la que se realizó la compra  Descripcion: de la compra registrada  Monto: de la compra | Códigos:  200 cuando es exitoso  400 si la tarjeta no se encuentra |
| TarjetaCredito | AgregarTarjtetaCliente | Agregar una tarjeta de crédito a un cliente | TarjetaCreditoDTO: información de la tarjeta de crédito que se le va a asignar al usuario | códigos: 200 cuando es exitoso  400 si la tarjeta no se encuentra  500 si hubo un error |
| TarjetaCredito | ObtenerListaTarjetas | Obtener una lista de tarjetas de crédito de todos los clientes | Sin parámetros ya que obtiene una lista de todas las tarjetas | códigos: 200 con la lista de tarjetas de crédito  500 si hubo un error |

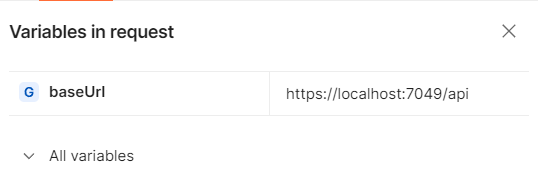
# Prueba de la solución

Correr el script de la base de datos (si es que aún no se ha ejecutado) para crear las tablas, procedimientos almacenados y la inserción de los datos de prueba las tablas correspondientes. Orden de ejecución del script:

* Los create table (cliente, tarjetaCredito y movimientoTarjeta)
* Los insert (cliente, tarjetaCredito y movimientoTarjeta)
* Ejecutar los procedimientos almacenados para crearlos

Con Postman:

En la variable global: baseUrl: cambiar la url por la dirección correcta de acuerdo con el servidor y el puerto que se tenga asignado, en mi caso es localhost y de puerto 7049

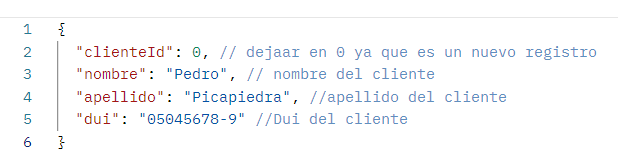


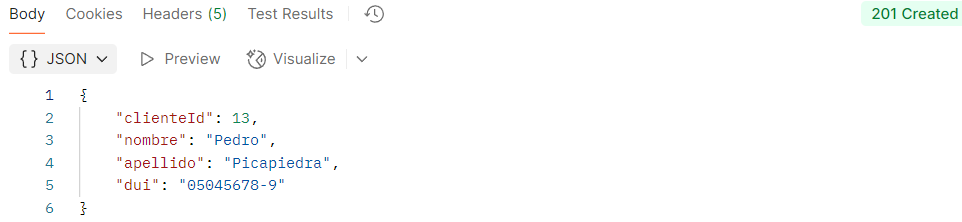
Verificar el estado con el health debería dar una respuesta como:

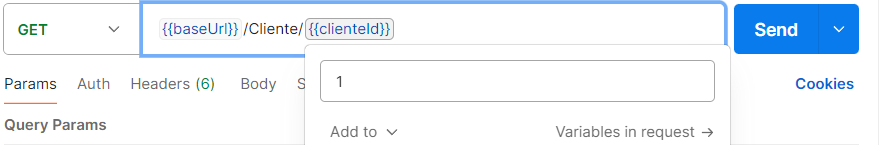


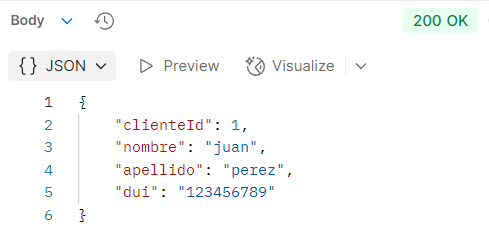
En la carpeta “cliente” hay 2 métodos un post para crear un cliente y un get para obtener información:

* Para crear: en el método de Post en la sección de Body la opción de raw y en el dropdown JSON ejemplo:

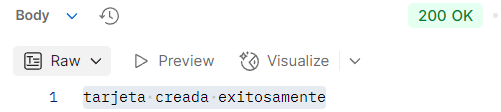
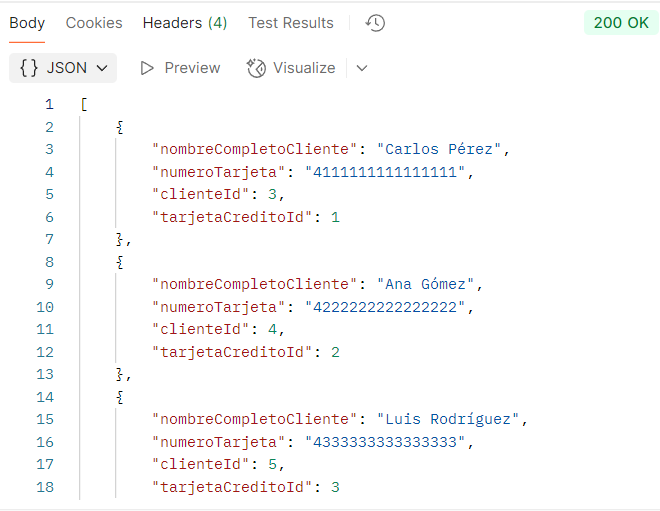


La respuesta seria  


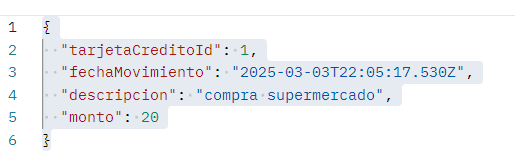
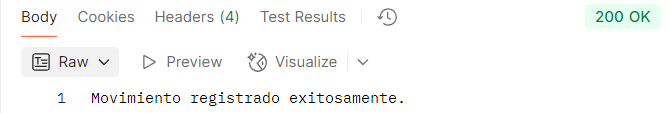
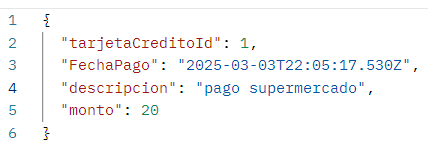
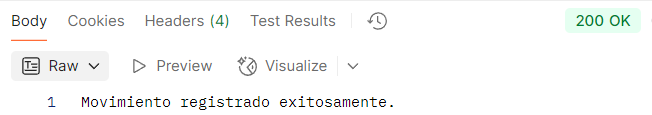
* En el “obtener cliente” es un método GET para hacer el get de otro id se cambiaria de la siguiente forma: pasar el cursor sobre la variable clienteId y se asigna un nuevo valor 

Con una respuesta:

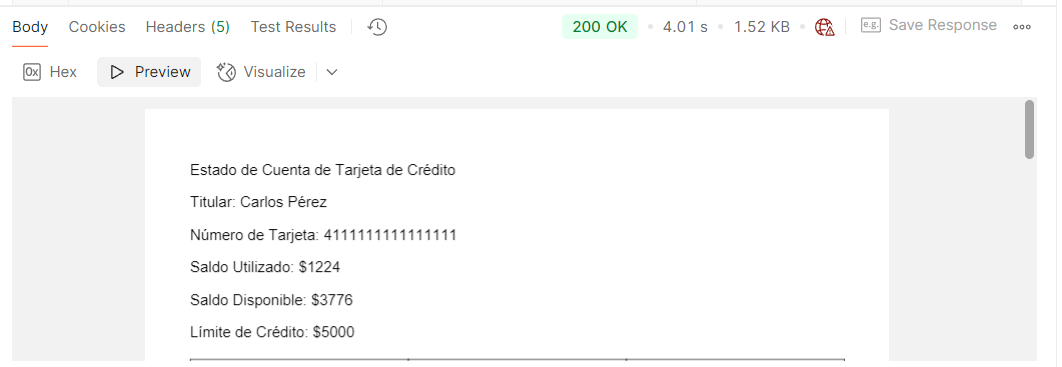
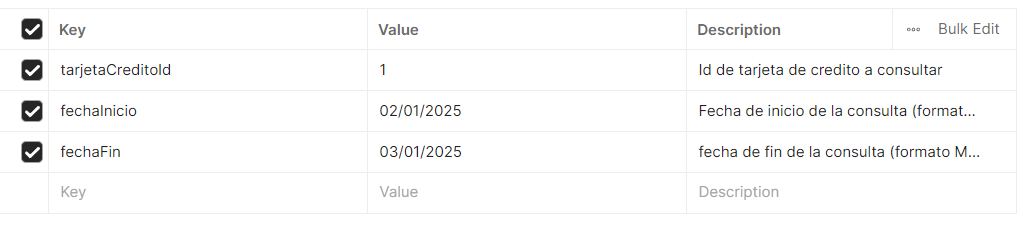
En la carpeta “Tarjeta de credito” hay 2 métodos un post para crear una relación entre cliente y tarjeta y un get para obtener el listado de tarjetas:

* Para crear: en el método de Post en la sección de Body la opción de raw y en el dropdown JSON ejemplo:  
    
  ejemplo de respuesta:   
  
* Este método get no necesita parámetros al ejecutarse daría un resultado como:  
  

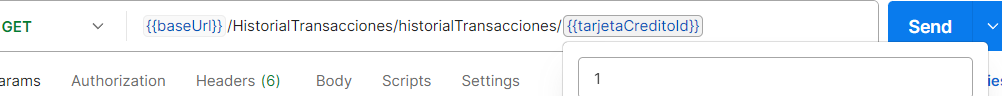
En la carpeta “Tarjeta de credito” hay 2 métodos post uno para crear un registro de compra y otro para registrar un pago:

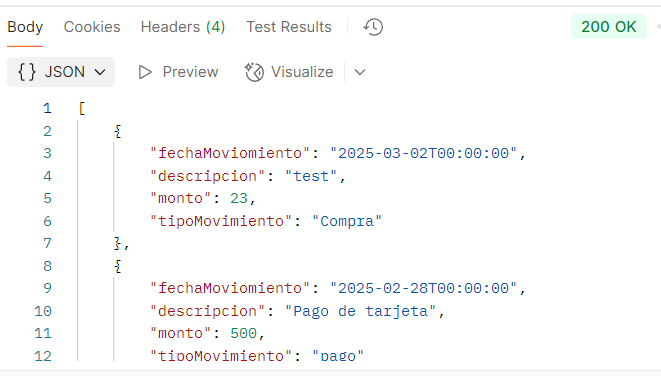
* Para registrar una compra (en el post compra): en el método de Post en la sección de Body la opción de raw y en el dropdown JSON ejemplo  
    
  y de resultado seria   
  
* Para registrar un pago (en el post pago): de la misma manera que la compra, en el Body   
    
  en respuesta seria:  
  

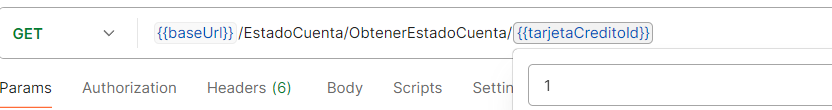
En la carpeta “exportar documentos” hay 2 métodos GET uno para exportar el PDF y uno para exportar Excel:

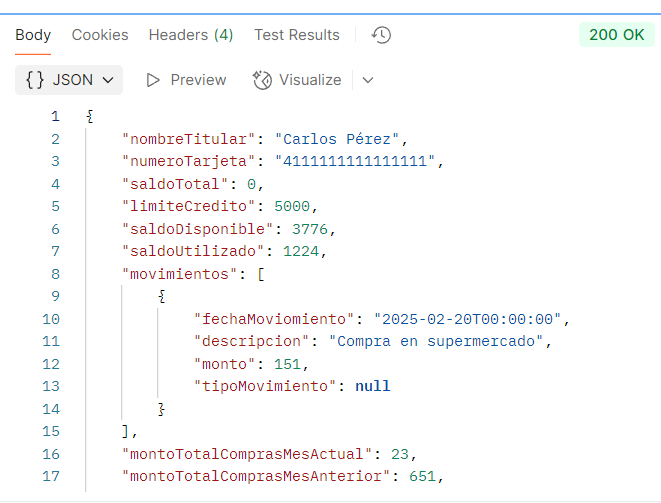
* En el “exportar pdf” es un método GET y para hacer el get de otro id se cambiaria de la siguiente forma: con pasar el cursor sobre la variable {{tarjetaCreditoId}} y se asigna un nuevo valor  
    
  y de resultado daría   
  
* En el “exportar Excel” es un método GET para asignar nuevos valores seria en la sección “Query Params”  
    
  como resultado 

En la carpeta “historial” hay un método GET: “obtener historial por tarjeta” para hacer el get de otro id se cambiaria de la siguiente forma: con pasar el cursor sobre la variable {{tarjetaCreditoId}} y se asigna un nuevo valor



Dando una respuesta:  
  
En la carpeta “estado de cuenta” hay un método GET: “obtener estado” para hacer el get de otro id se cambiaria de la siguiente forma: con pasar el cursor sobre la variable {{tarjetaCreditoId}} y se asigna un nuevo valor



En respuesta tendríamos:  


La estructura general esta:   
