Prueba Técnica

Autor: Christofer Jhonatan Palacios Villegas

Nombre de la solución: TarjetaCPruebaAPI

Descripción General:

El proyecto tiene como funcionalidad el poder realizar y ver los movimientos de un usuario que posea una tarjeta de crédito, donde pueda ver sus transacciones realizadas, sus compras y sus abonos a tarjeta de crédito, además de ver su estado de cuenta. El proyecto tiene como arquitectura principal Blazor Server, el cual consume una API REST utilizando .NET y un motor de base de datos SQL Server.

Estructura del Proyecto:

La solución contiene dos proyectos utilizando Blazor Server, uno llamado TarjetaCPruebaAPI y el otro llamado TarjetaCPrueba, donde el primer proyecto es donde se definió la lógica del programa y donde se configuro la cadena de conexión de base de datos (appsettings.json), la capa de acceso a datos y un filtro de excepciones personalizado (Program.cs). Se puede observar el conjunto de carpetas (Controllers, Data, Filters, Models) en donde Models se encuentran los objetos de base de datos, en Data se encuentra el contexto a la base de datos, en Filters un código que implementa un filtro de excepciones para la API, y en Controllers tenemos los métodos CRUD para el programa, donde se puede localizar los Endpoints utilizados para cada controlador.

Tecnologías Utilizadas:

Para este proyecto se utilizo el lenguaje de programación C# ya que el ecosistema principal es .NET, teniendo una sintaxis limpia permitiendo un desarrollo estructurado y escalable. El proyecto a nivel de front end esta basado en Blazor Server App, ya que facilita el desarrollo de aplicaciones web modernas con la lógica del lado del servidor y permitiendo código C# nativo en la definición de la interfaz gráfica, contando también con .NET 6.0, ya que es la versión más moderna y robusta del framework haciendo al proyecto confiable y escalable para las aplicaciones web. Por otro lado tenemos Entity Framework Core que nos ha permitido la ligeres y la eficiencia para simplificar el acceso y la manipulación de datos en base de datos relacionales desde la aplicación .NET y para el motor de base de datos se utilizo SQL Server ya que nos garantiza una integración profunda y optimizada entre la aplicación .NET, ambas desarrolladas por Microsoft teniendo la ventaja de que ofrece soporte y actualizaciones continuas para el motor de base de datos garantizando estabilidad y seguridad al sistema.

Nombre del controlador	Endpoint	Método HTTP	Descripción
CreditCardsController	GET /api/CreditCards	GET	Obtiene una lista de todas las tarjetas de crédito.
	GET /api/CreditCards/{id}	GET	Obtiene una tarjeta de crédito específica por su ID.
	POST /api/CreditCards	POST	Crea una nueva tarjeta de crédito.
	PUT /api/CreditCards/{id}	PUT	Actualiza una tarjeta de crédito existente por su ID.
	DELETE /api/CreditCards/{id}	DELETE	Elimina una tarjeta de crédito por su ID.
TransactionsController	GET /api/Transactions	GET	Obtiene una lista de todas las transacciones.
	GET /api/Transactions/{id}	GET	Obtiene una transacción específica por su ID.
	POST /api/Transactions/SaveTransaction	POST	Guarda una nueva transacción.
	PUT /api/Transactions/{id}	PUT	Actualiza una transacción existente por su ID.
	DELETE /api/Transactions/{id}	DELETE	Elimina una transacción por su ID.
	GET /api/Transactions/card/{cardId}	GET	Obtiene todas las transacciones asociadas a una tarjeta

			específica por su ID.
UsersController	GET /api/Users	GET	Obtiene una lista de todos los usuarios.
	GET /api/Users/{id}	GET	Obtiene un usuario específico por su ID.
	POST /api/Users	POST	Crea un nuevo usuario.
	PUT /api/Users/{id}	PUT	Actualiza un usuario existente por su ID.
	DELETE /api/Users/{id}	DELETE	Elimina un usuario por su ID.
	GET /api/Users/userswithcreditcards	GET	Obtiene una lista de usuarios con sus respectivas tarjetas de crédito.
	GET /api/Users/userStatement?userId={userId}	GET	Obtiene el estado de cuenta de un usuario específico, incluyendo detalles de transacciones asociadas a su tarjeta de crédito.

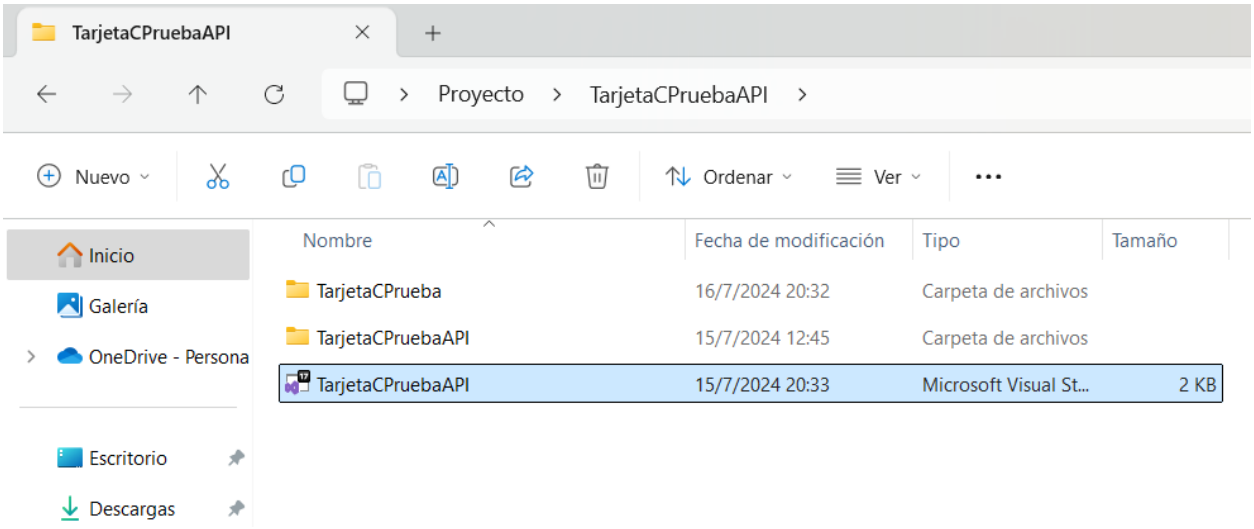
Consideraciones:

Cada controlador (CreditCardsController, TransactionsController, UsersController) define múltiples endpoints que corresponden a operaciones CRUD básicas (Create, Read, Update, Delete) y consultas adicionales según el contexto (como obtener transacciones por tarjeta o usuarios con tarjetas de crédito).

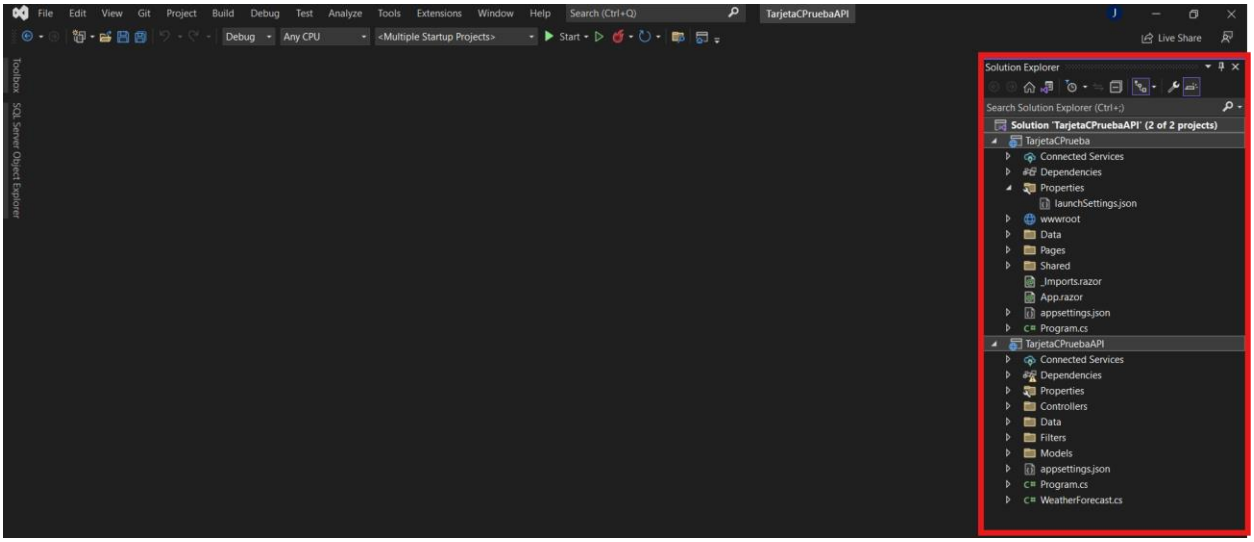
Estos endpoints forman la base que interactuar con los datos de la aplicación a través de una API RESTful, proporcionando operaciones estándar y consultas personalizadas para gestionar tarjetas de crédito, transacciones y usuarios en tu sistema.

Como Probar La Aplicación

1- Descargar el proyecto proporcionado desde GitHub(link)
2- Abrir la carpeta descargada y dentro de ella se encontrará la solución para abrir con Visual Studio la cual trae 2 proyectos (TarjetasCPruebaAPI y TarjetasCPrueba)

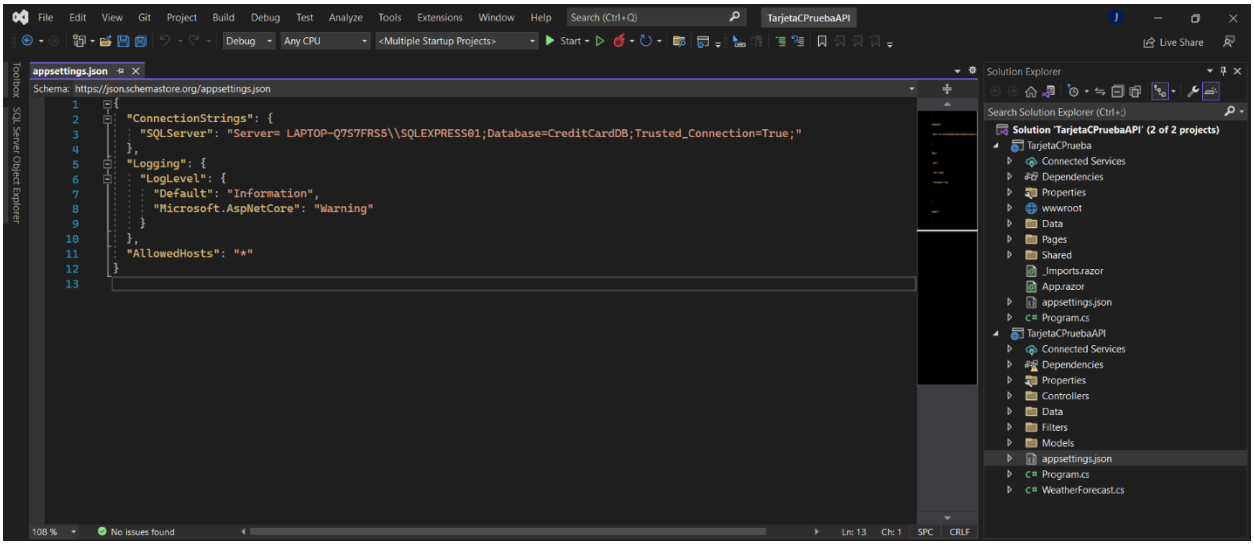


3- Una vez dentro de la solucion podremos observar la estructura de los dos proyectos.



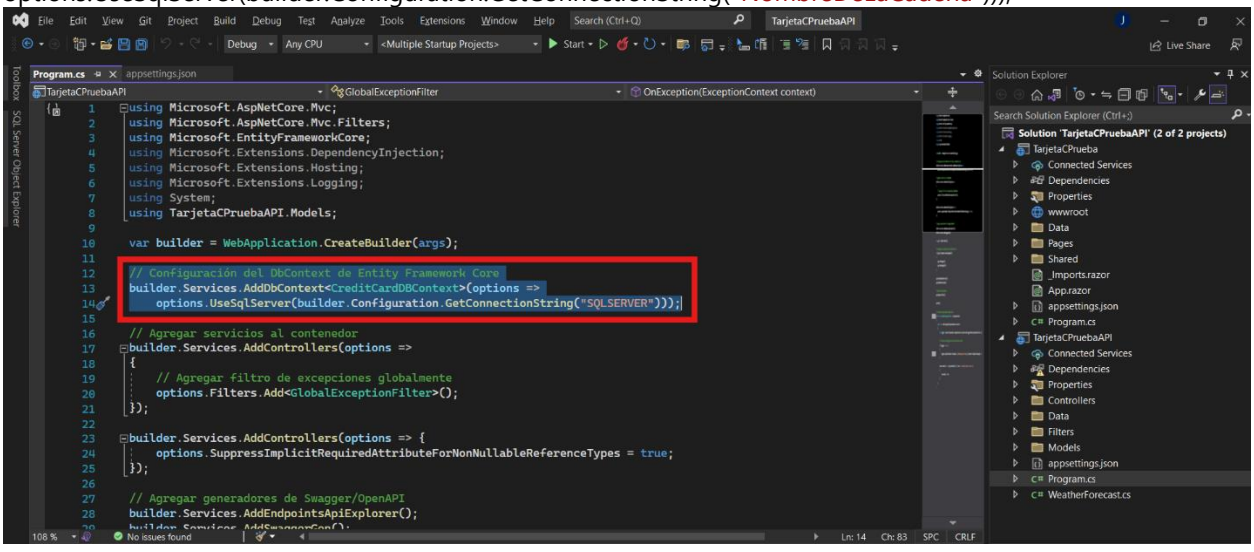
4- Configurar la cadena de conexión a la base de datos, la cual se encuentra en TarjetasCPruebaAPI-appsettings.json. Ejemplo para copiar en su código:

```
{
  "ConnectionStrings": {
    "NombreDeLaCadena": "Server=
nombre_del_servidorSQL;Database=CreditCardDB;Trusted_Connection=True;"
  },
```

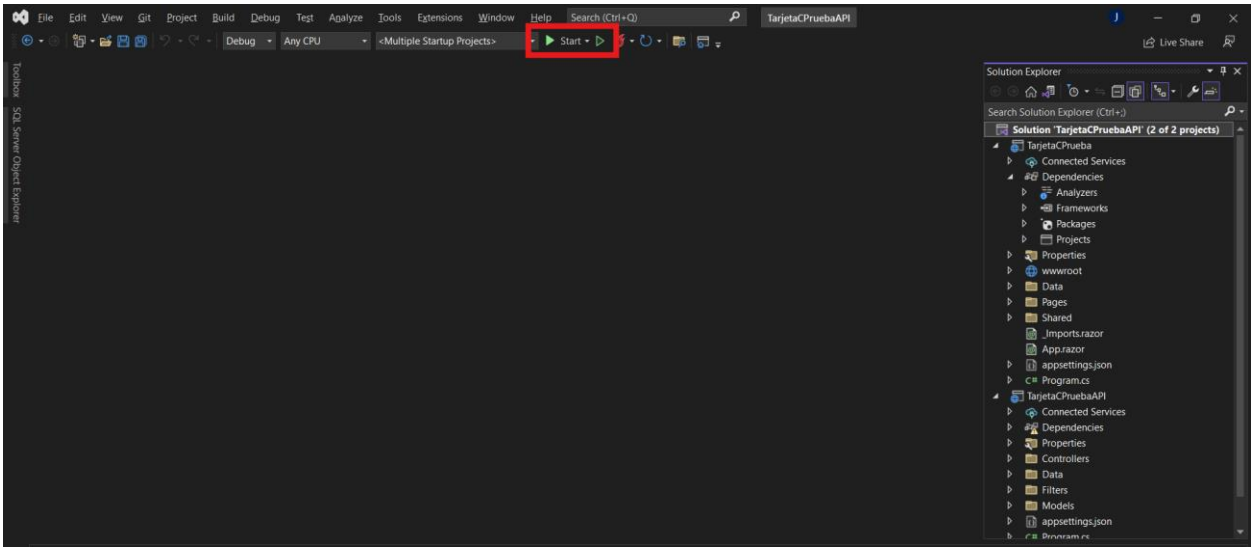


5- Configurar los servicios en TarjetasCPruebaAPI-Program.cs. Ejemplo para copiar en su código:

```
// Configuración del DbContext de Entity Framework Core
builder.Services.AddDbContext<CreditCardDBContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("NombreDeLaCadena")));
```

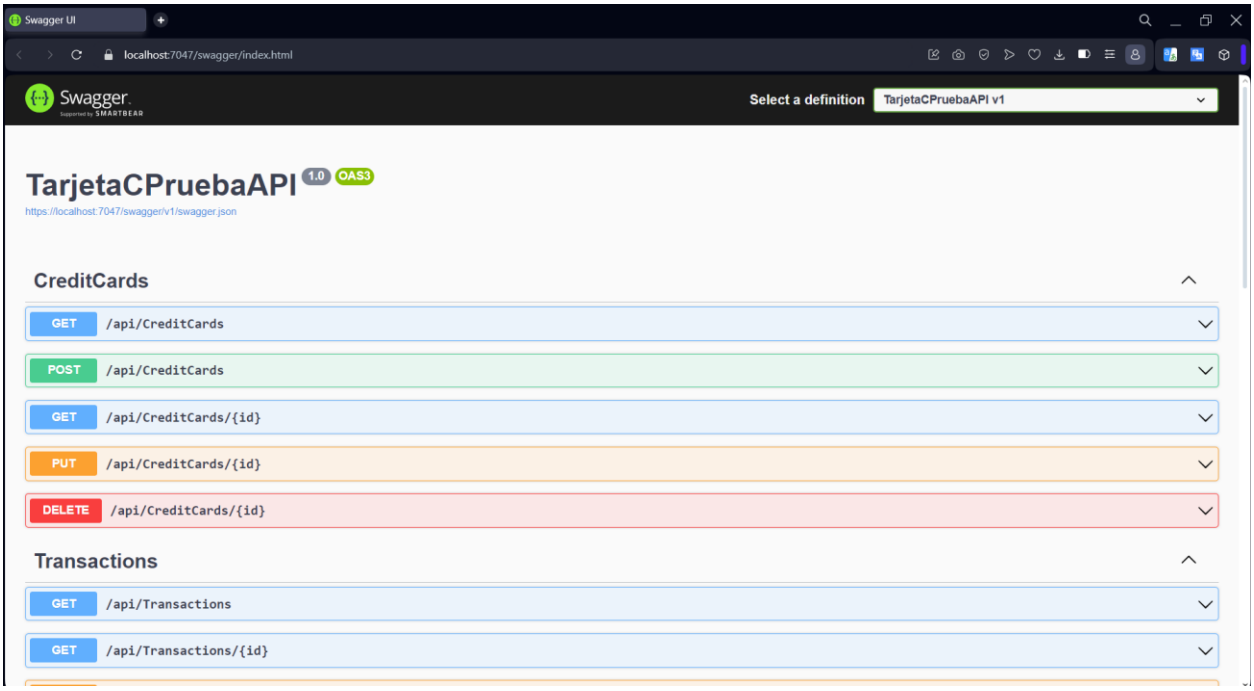


6- Una vez configurada la conexión, procederemos a ejecutar la aplicación para poder visualizar el funcionamiento de la misma, para ello buscamos el botón de Start en la parte superior de Visual Studio, tal como se observa en la siguiente imagen.

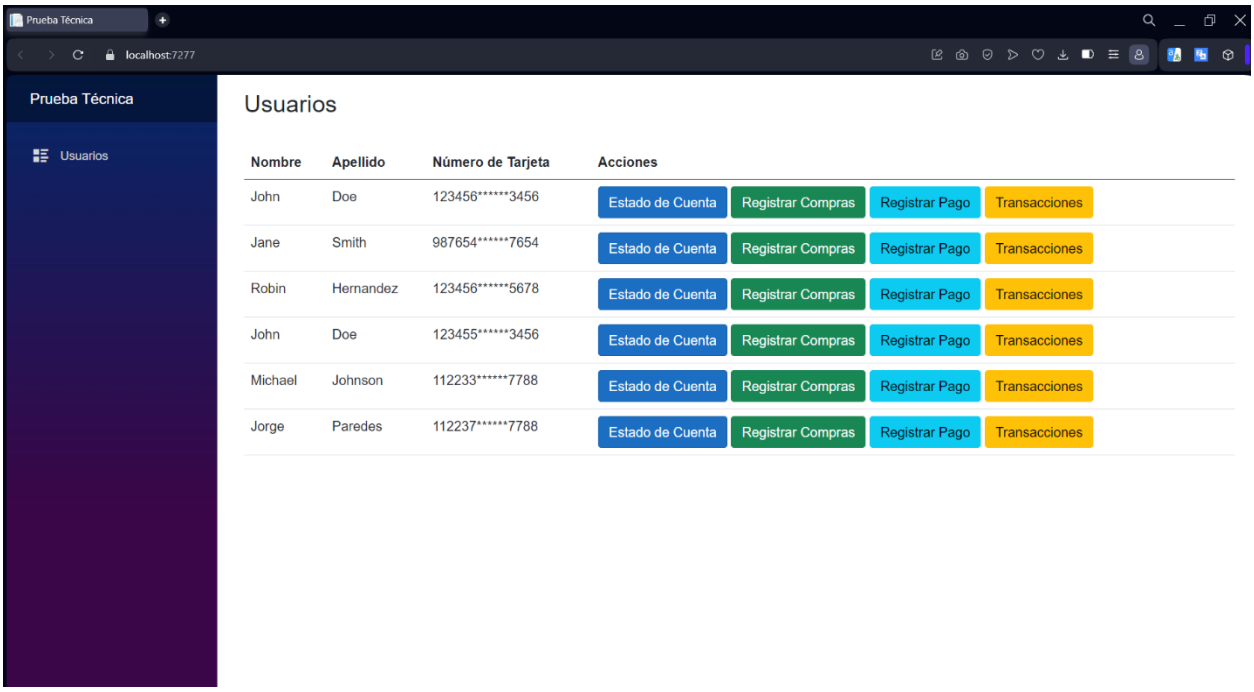


7- Al momento de ejecutar la aplicación se abrirá nuestro navegador predeterminado con dos pestanas, en una podremos observar el Swagger y en la otra pestana podemos observar nuestra aplicación, la que por defecto cargará nuestros usuarios que contengan tarjeta de crédito.

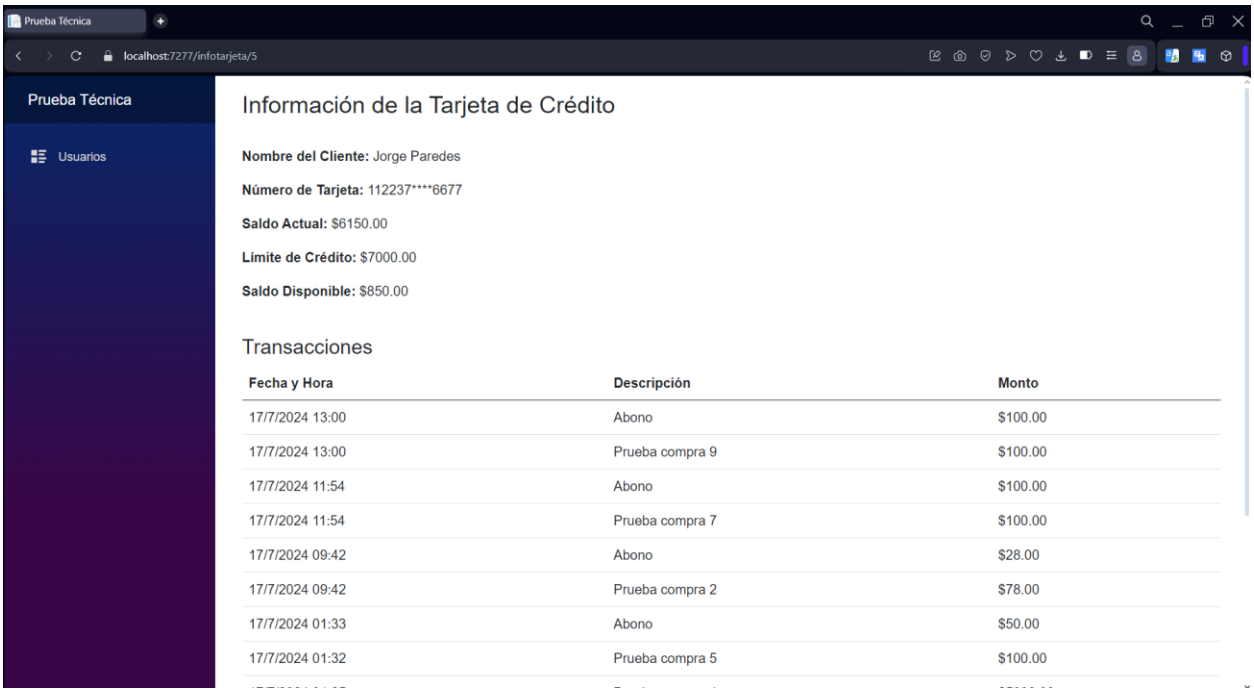
Pagina inicial de Swagger:



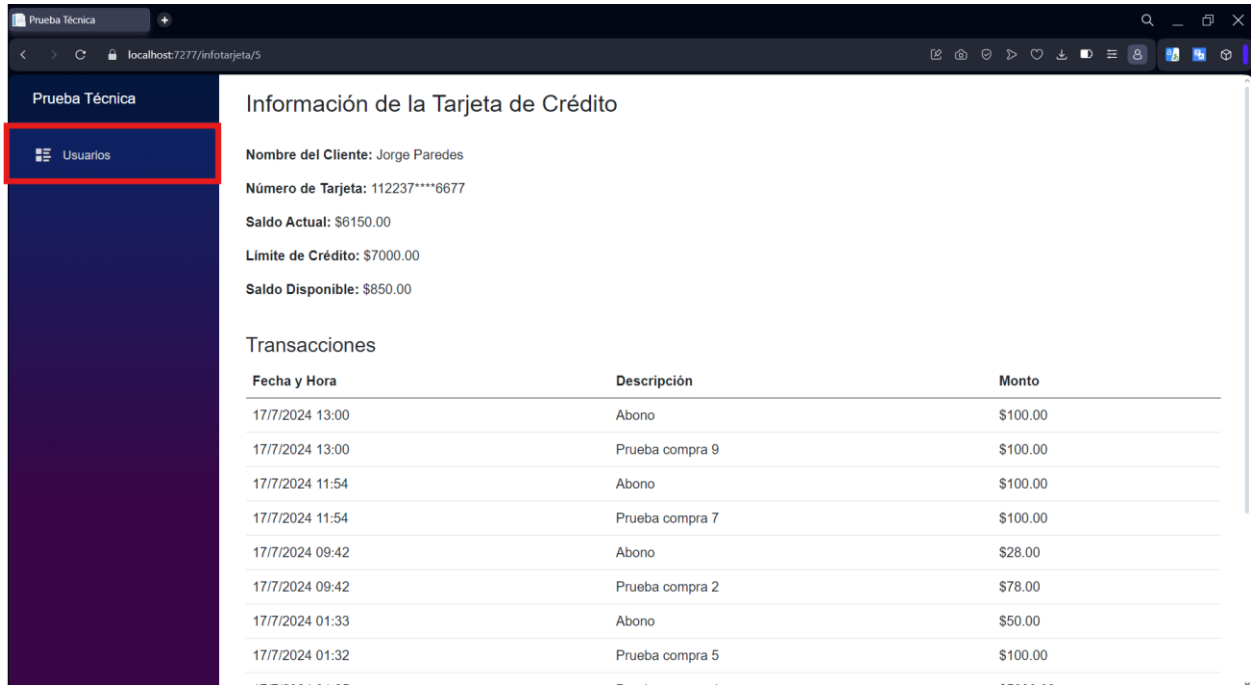
Pagina inicial de nuestra aplicación:



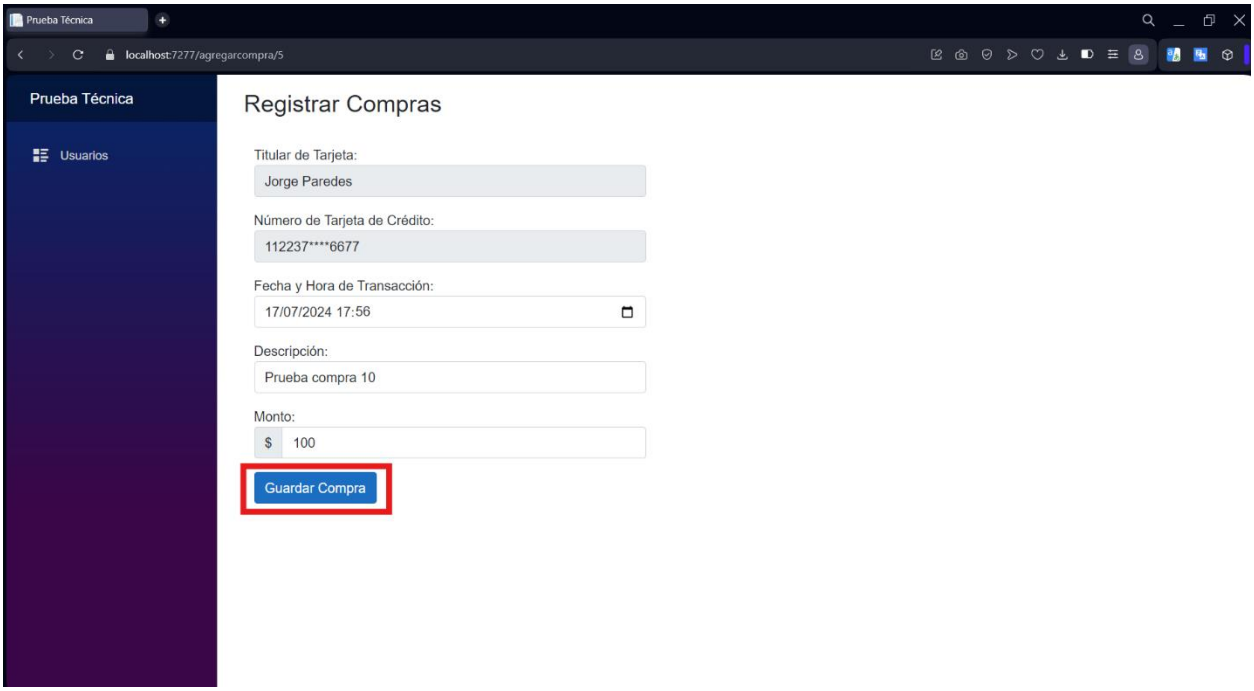
8- Para comprobar el funcionamiento de nuestra aplicación, tomaremos como base al usuario Jorge Paredes, para esto observaremos su Estado de Cuenta, para luego insertar compras y abonos y así ver como los datos cambian según lo que el cliente haga.



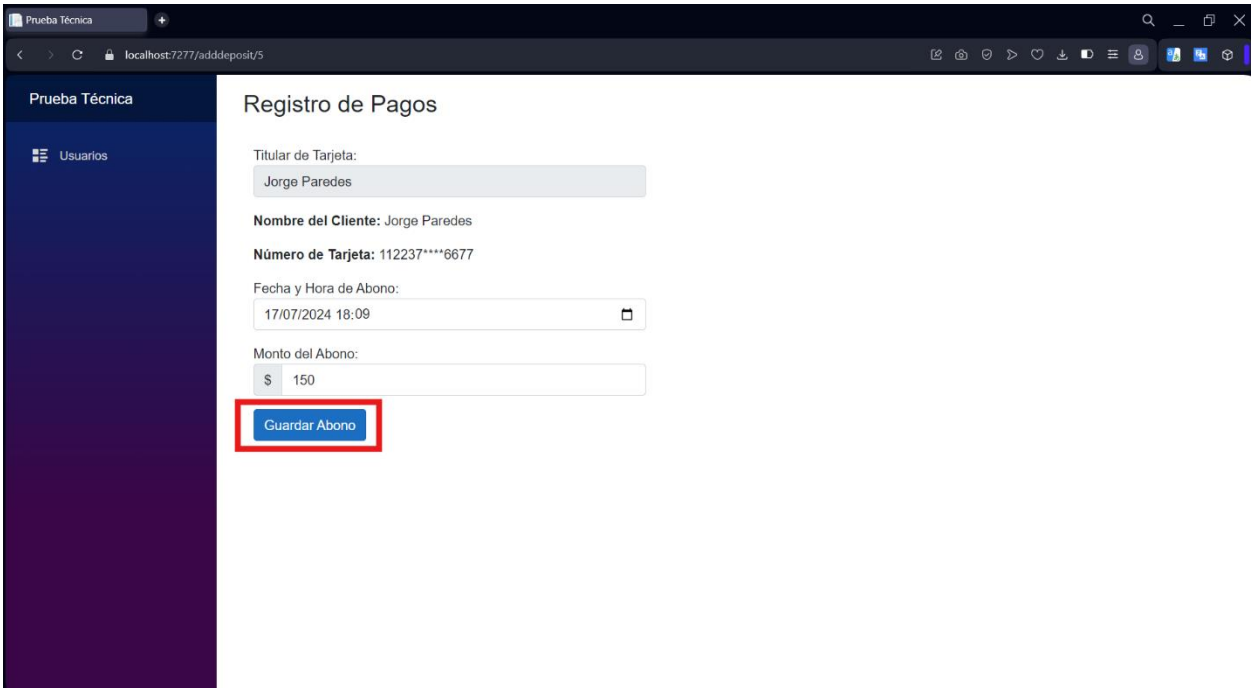
9- Una vez observado los datos del usuario, regresamos al menú principal, para esto buscamos en la barra izquierda, la opción de usuarios y presionamos en Usuarios, tal como se muestra en la imagen:



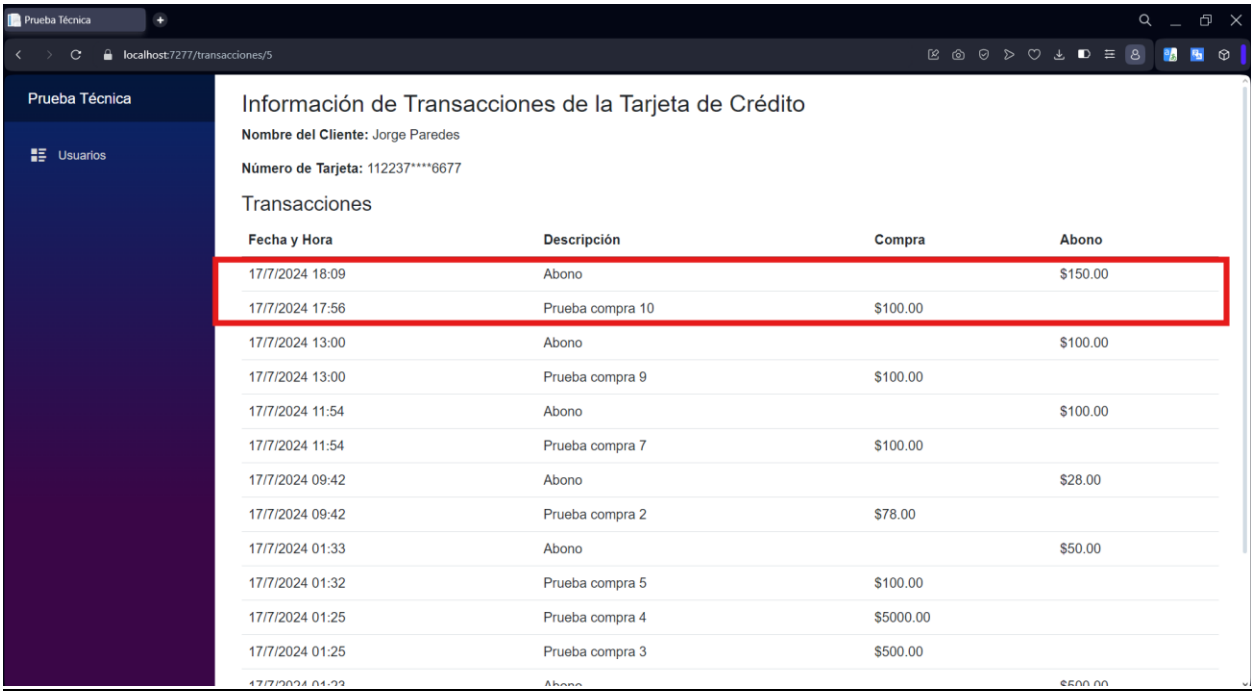
10- Estando de regreso en el menú registramos una compra para el usuario Jorge Paredes en Registrar Compras, llenamos los campos que se pide llenar y guardamos la compra con Guardar Compra, luego de registrar la compra nos devolverá al menú principal Usuarios.



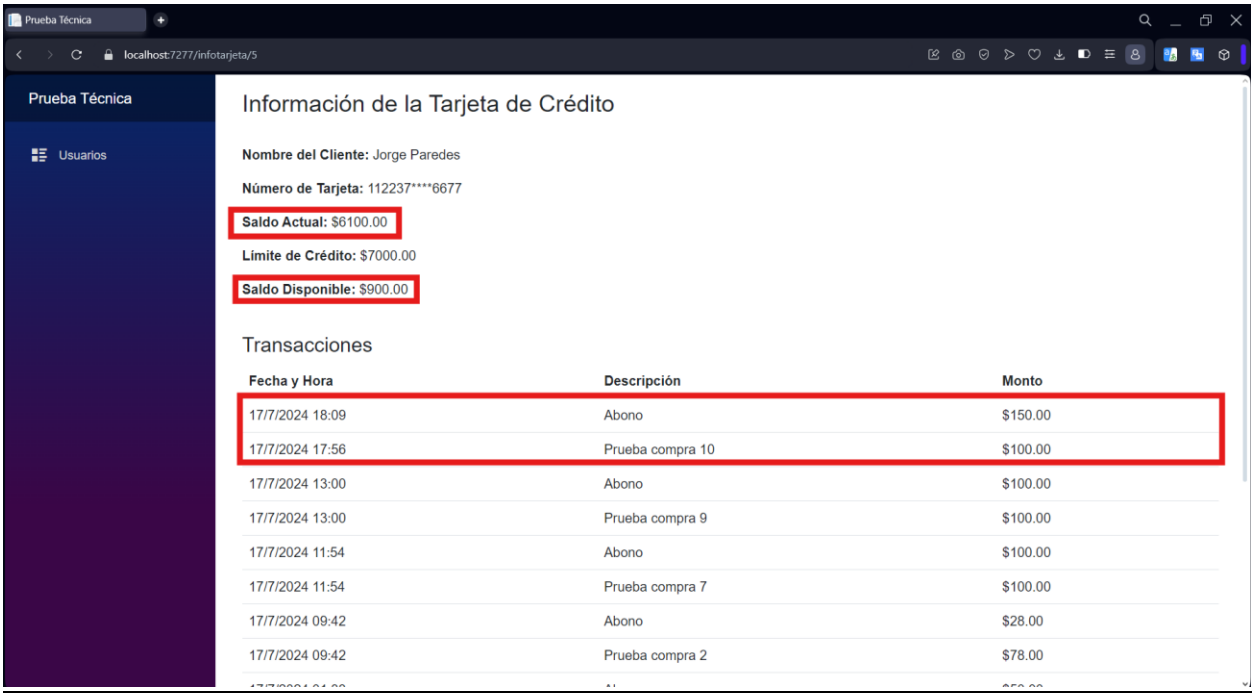
11- Estando de regreso en el menú principal de Usuarios, podemos ir a realizar un abono a la tarjeta en Registrar Pago, llenamos los espacios correspondientes y damos clic en el botón de Guardar Abono, llevándonos nuevamente al menú principal de Usuarios.



12- Ahora podremos observar en Transacciones las transacciones que se han realizado en este proceso con su respectiva fecha y la hora en la que se realizó la transacción.



13- Nuevamente nos movilizamos al menú de Usuarios, en la barra izquierda, e ingresamos a Estado de Cuenta, donde observaremos nuestras transacciones realizadas y como estas han afectado en Saldo Actual, y Saldo Disponible.



Información Extra:

El código tiene validaciones, una de ellas es que, si el usuario tiene X límite de crédito, este no le permitirá hacer compras mayores a ese crédito que posee el usuario. También tiene la validación de que al momento de querer agregar una compra o abonar a la tarjeta, no permitirá números negativos.