PONTIFICA UNIVERSIDAD CATÓLICA DEL ECUADOR

Pupiales Quinatoa Javier 2024-06-05

Integración de sistemas

Ing. Damián Nicolade

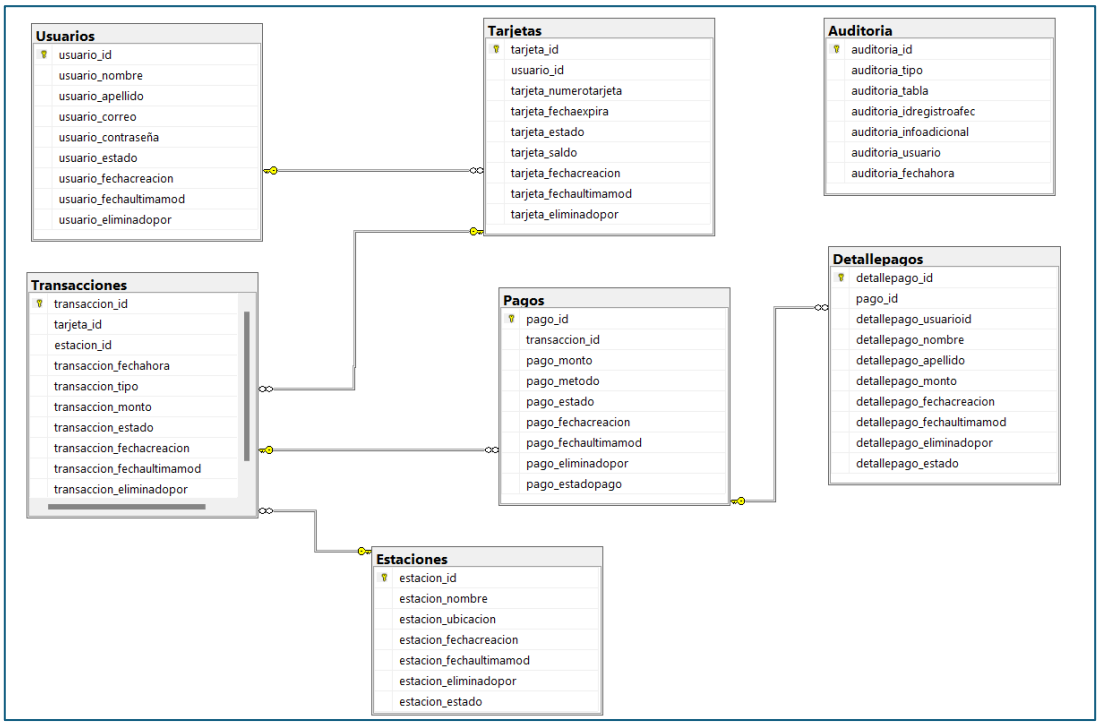
Pruebas unitarias

SISTEMA TRANSACCIONAL DEL METRO DE QUITO

El sistema transaccional del metro de Quito tiene como objetivo, trabajar con las tablas del sistema desde una interfaz amigable para los trabajadores del metro de Quito. A nivel macro, este sistema se maneja con 7 tablas.

- Usuarios
- Tarjetas
- Transacciones
- Estaciones
- Pagos
- Detalle Pagos
- Auditoria

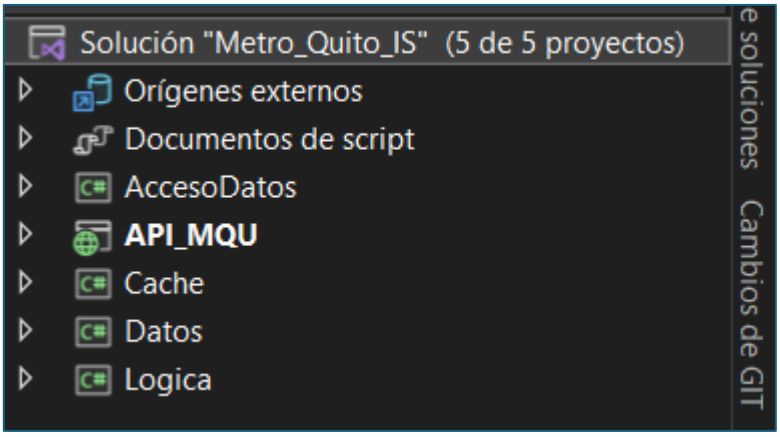
Modelo de base de datos



Cada tabla tiene sus métodos CRUD (Create, Read, Update, Delete), lo que ayuda a ingresar nuevos clientes, dar mantenimiento a transacciones, generar pagos y órdenes de pago.

La lógica de gestión del sistema del Metro de Quito se realizo en Visual Studio code en una programación en N capas teniendo:

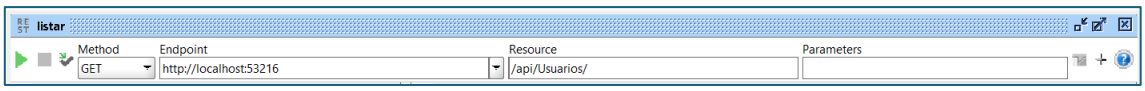
- Acceso Datos
- Datos
- Cache
- Lógica
- API

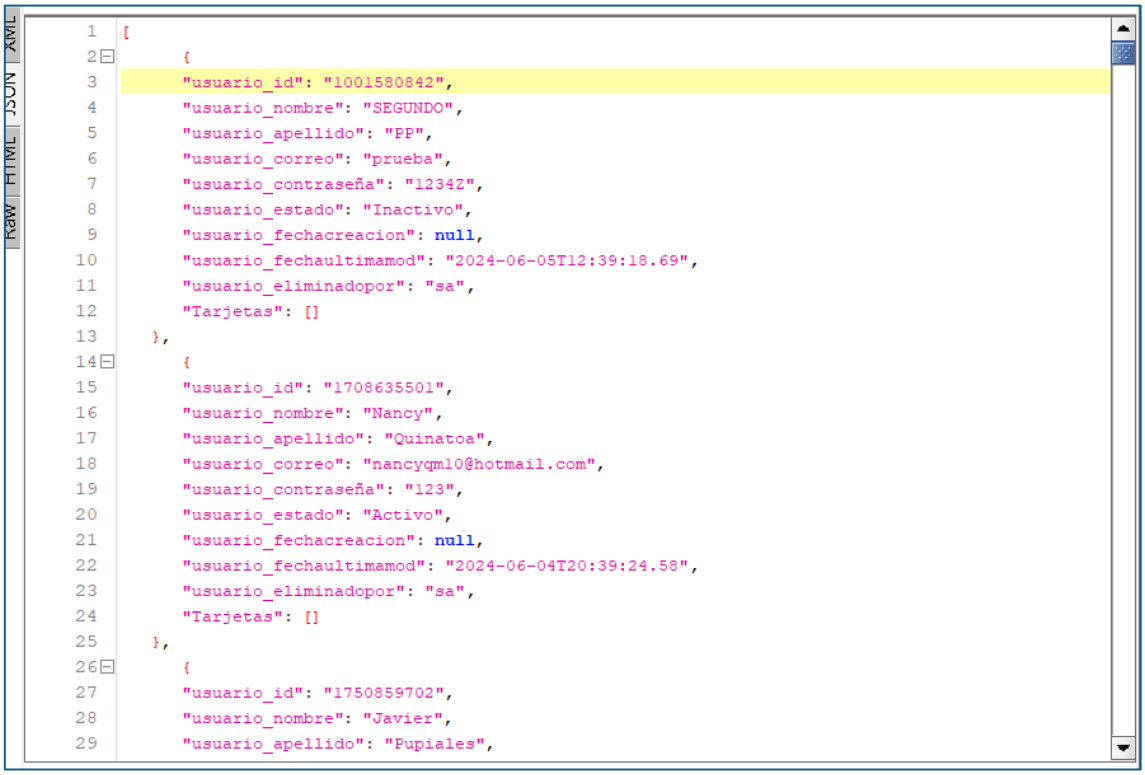


Para sacar a producción este proyecto, el sistema debe pasar por varias pruebas para mitigar errores en el ambiente de producción. A continuación, se presentan las pruebas que se realizo desde SOAP UI para las llamadas a la API del sistema, adicionalmente se realizara la prueba de integración con la banca web para ver el funcionamiento del bus de datos y como el sistema interactúa con sistemas externos.

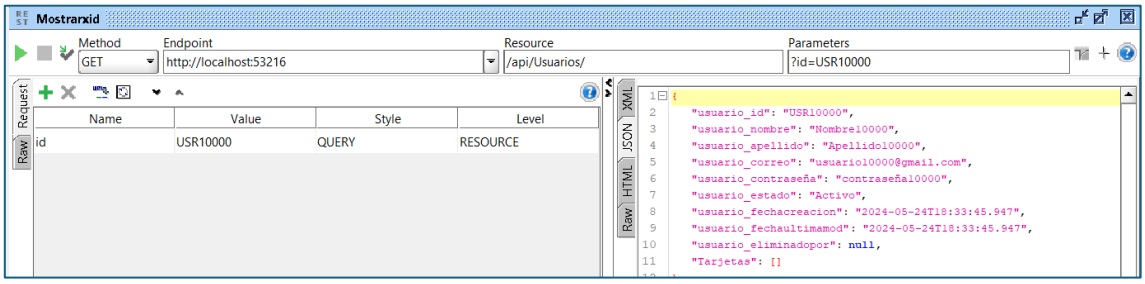
PRUEBAS UNITARIAS

LISTAR CLIENTES

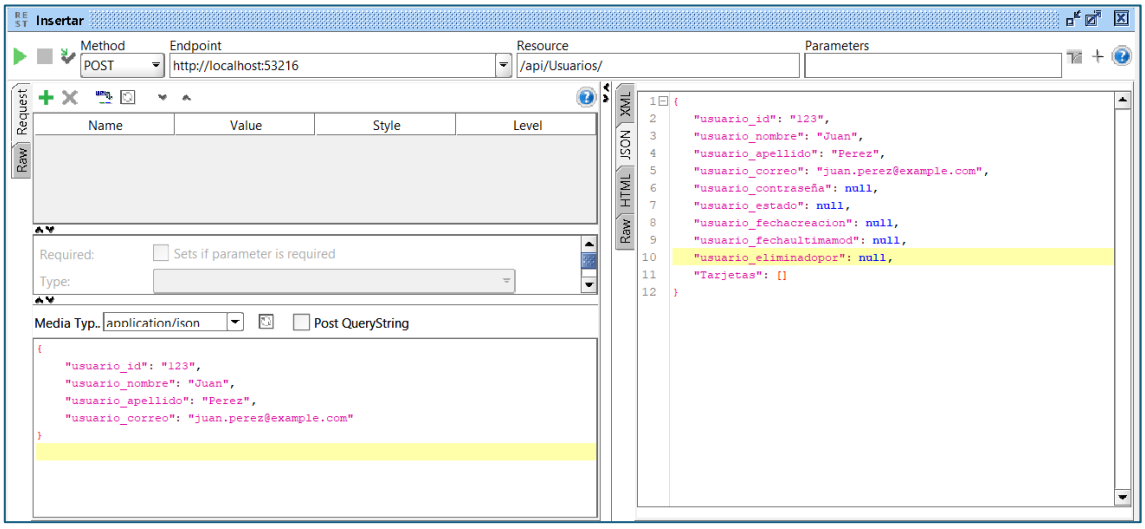




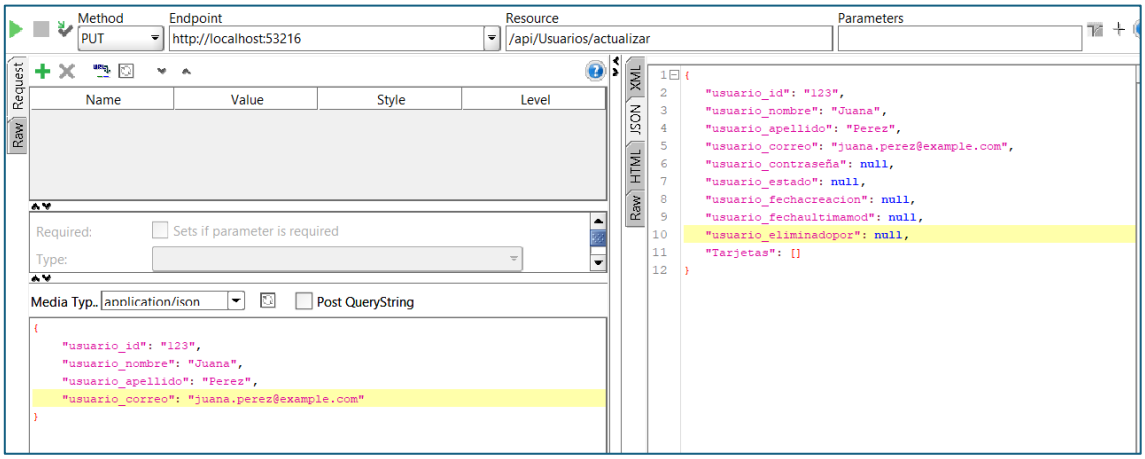
Ver clientes por ID



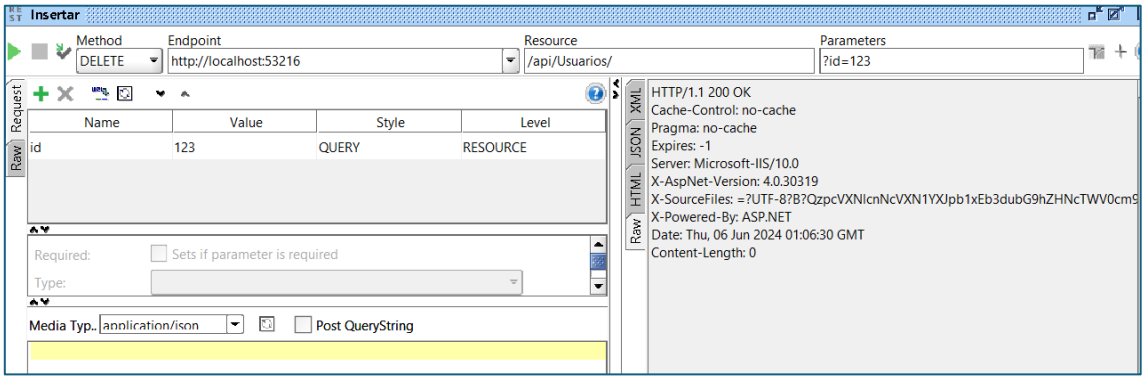
Crear clientes



Actualizar Cliente:

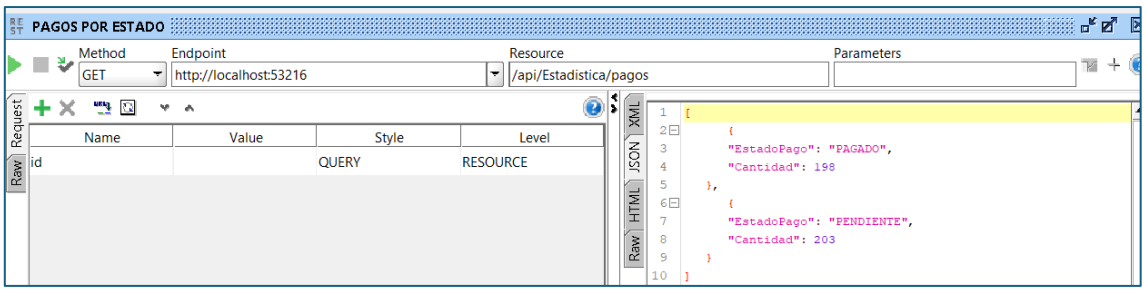


Eliminar un cliente

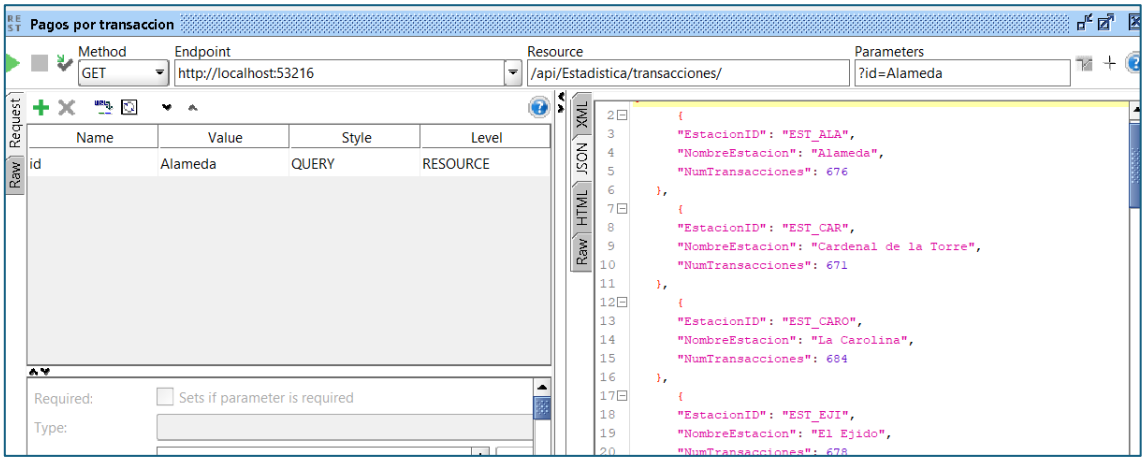


Consulta a SPS

Pagos por estado



Transacciones por estación



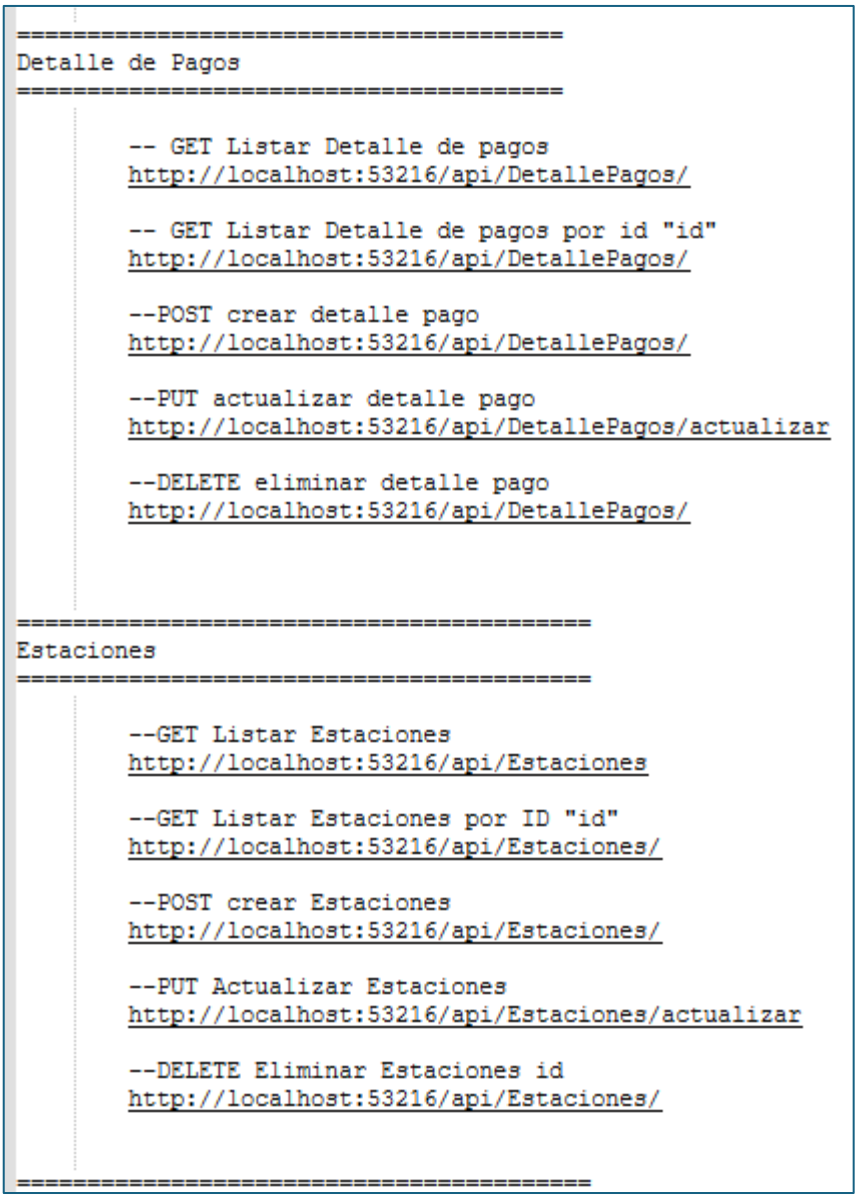
Pruebas desde la interfaz

Una vez se prueba que las apis funciona, se debe toma run diccionario de URIS, indicando todas las URIS que se tienen en el sistema de metro de Quito:

```
=========================================
Auditoria
=========================================
        --GET Listar
        http://localhost:53216/api/Auditoria/

        -- GET Listar por id "id"
        http://localhost:53216/api/Auditoria/

=========================================
Busqueda transacciones por estacion "estacion"
=========================================
        --GET
        http://localhost:53216/api/Busqueda/
```



FRONT END:

Al momento de implementar el front-end se debe continuar con las pruebas unitarias, ya que puede que algo no funcione bien, una petición no se mande bien o no se esta recibiendo la información. Para esto es buena practica tener manejo de excepciones en cada capa dentro de visual estudio, dentro de la programación con javascript en el frontend, también podemos tener variables de console log para ver que se estén recibiendo los datos o problemas con la interfaz.

ConsoleLog con JavaScript:

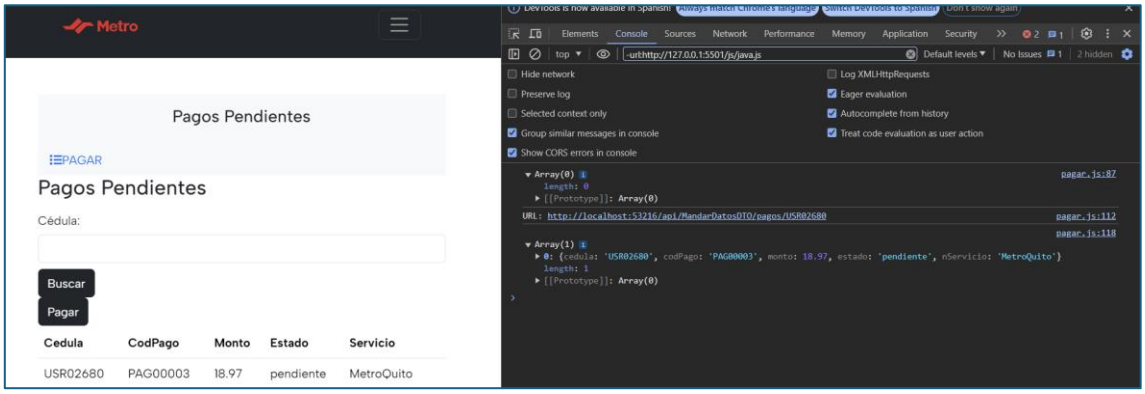
```
function getClienteById() {
    var id = $('#searchId').val();
    if (!id) {
        getClientes();
        return;
    }

    // Construir la URL y registrarla en la consola para verificación
    var url = 'http://localhost:53216/api/MandarDatosDTO/pagos/' + encodeURIComponent(id);
    console.log("URL: " + url); // Verificar la URL generada

    $.ajax({
        url: url,
        type: 'GET',
        success: function (data) {
            console.log(data); // Añade esto para ver los datos en la consola
            currentClientes = Array.isArray(data) ? data : [data];
            if (data.length) {
                displayClientes(1);
                $('#errorMessage').hide();
            } else {
                $('#errorMessage').show().text('No hay registros');
            }
        },
        error: function (xhr, status, error) {
            console.error('Error al obtener datos:', xhr.status, xhr.statusText, xhr.responseText
            alert('Error al obtener datos: ' + xhr.status + ' ' + xhr.statusText + ' ' + xhr.resp
        }
    });

    // Limpiar el valor del input
    $('#searchId').val("");
}
```

Con herramientas de desarrollador de los navegadores web se pueden ver los resultados del ConsoleLog



Manejo de excepciones desde visual studio:

En caso de que no se encuentren errores desde el front y surge algún problema durante el envió de la petición podemos ejecutar una depuración dentro de visual estudio, poniendo un punto de interrupción en el método que se este intentando llamar y no responda de la manera esperada

```
47          }
48          // POST api/Pagos
49          [HttpPost]
            0 referencias
50          public IHttpActionResult Post([FromBody] Pagos nuevoPago)
51          {
52              if (nuevoPago == null)
53              {
54                  return BadRequest("Datos inválidos");
55              }
56
57              try
58              {
59                  APIPagos.Insertar(nuevoPago);
60                  return CreatedAtRoute("DefaultApi", new { id = nuevoPago.pago_id }, nuevoPago);
61              }
62              catch (Exception ex)
63              {
```

Gracias a estas depuraciones se puede ver donde se cayo la llamada, con esta información podremos ir mas a detalle al problema, en este caso se pudo detectar gracias al manejo de excepciones por capas y la depuración paso a paso, un problema con un trigger en la base de datos al momento de guardar los Logs de Auditoria

```
INSERT INTO BD_Metro_Quito.dbo.Pagos (
    pago_id,
    transaccion_id,
    pago_monto,
    pago_metodo,
    pago_estadopago
) VALUES (
    'PAGOPRUEBA', -- Asigna el ID del pago
    'TRXPRUEBA', -- Asigna el ID de la transacción correspondiente
    23, -- Asigna el monto del pago
    'EFECTIVO', -- Asigna el método de pagoAsigna el usuario que eliminó el pago (si aplica)
    'PAGADO' -- Asigna el estado del pago
);
```

```
0 %
Mensajes

(1 fila afectada)

(1 fila afectada)

(1 fila afectada)
Mensaje 8115, nivel 16, estado 2, procedimiento tr_InsertAudit_Pagos, línea 16 [línea de inicio de lote 0]
Error de desbordamiento aritmético al convertir expression al tipo de datos nvarchar.
Se terminó la instrucción.

Hora de finalización: 2024-06-05T01:08:23.8026979-05:00
```

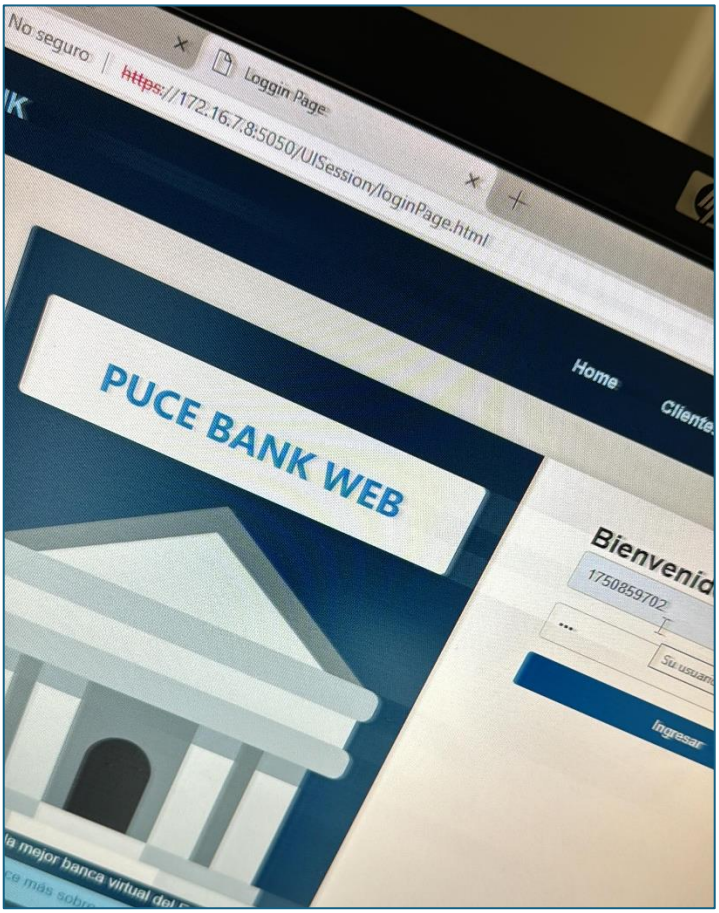
PRUEBAS DE INTEGRACION:

Una vez se realice las pruebas también mandando peticiones desde el frontend se puede pasar a interactuar con sistemas externos, en este caso con PUCEBANK

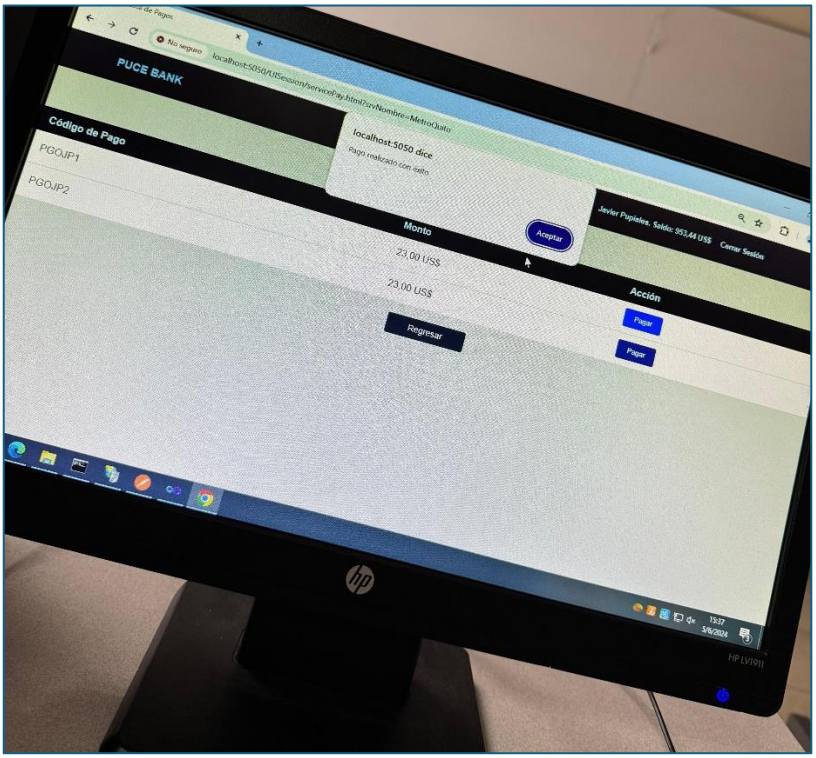
Lo primero que debemos hacer es generar Pagos Pendientes en el sistema del metro de quito

Cuando se da al botón de Pagar se redirige a la Banca Web

Se inicia Sesión a la cuenta asignada en la banca Web con el numero de cedula



Se ve los pagos pendientes y se cancela el registro pendiente de pago



Se verifica que se haya actualizado el pago en el sistema de Metro de Quito