SISTRAN

Manual Tecnico WebApi Nuvem

Título: Manual tecnico WebApi Nuvem
Compañía: Axa Colpatria
Proyecto: Nuvem
Revisiones: 1

Fecha	Versión	Autor	Comentario	Estado	Aprobación	
					Fecha	Aprobado por
06/05/2019	1	Jorge David Pertuz Egea	Consumo de servicios RestFull	Inicio		

Tabla de Contenido.

1 Introducción

1.1 Propósito

Este documento tiene como propósito principal, explicar cuál es la forma en la que se realizo la webapi que expune los servicios Soap de AXA en formato Rest.

No es del alcance de este documento la administración del WebApi en el servidor ya que esto dependerá de las áreas que lo administren.

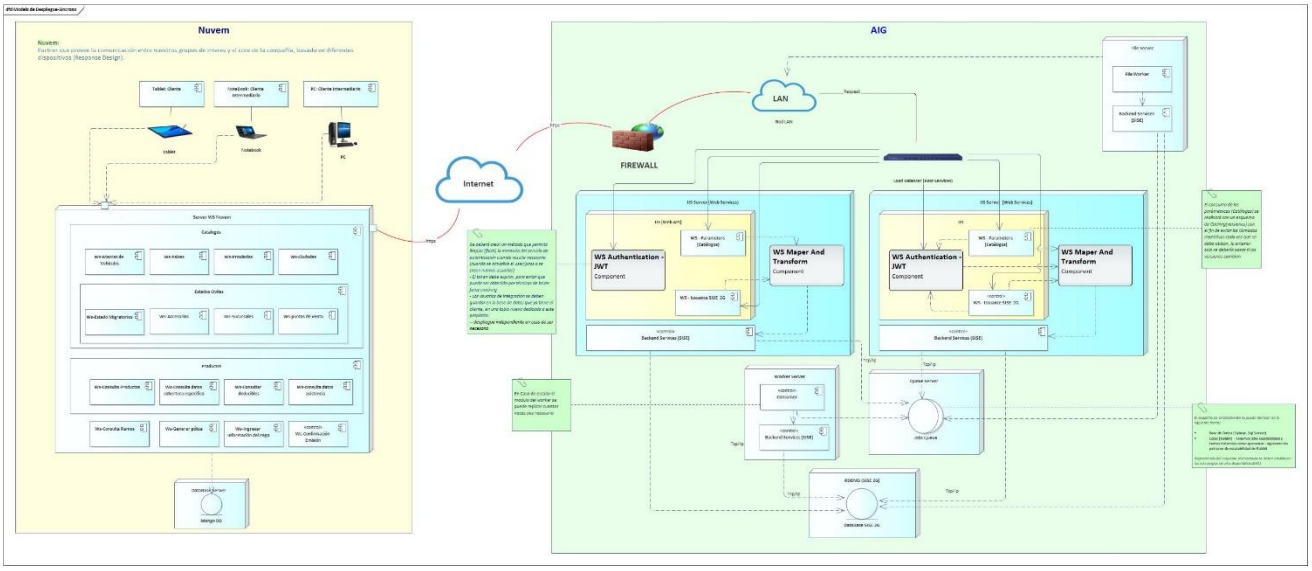
2 Supuestos

Dentro de este manual se supone como mínimo que la máquina donde se realiza la instalación contiene una arquitectura de 64 bits, así como un sistema operativo instalado de acuerdo con esta arquitectura. Mínimo 4 GB en memoria RAM, junto a Framework .Net Core 2.0.

Debe existir un servidor con los servicios Soap expuestos, para ser consumidos por la WebApi.

3 Arquitectura del WebApi

La arquitectura planteada para esta version de prueba del WebApi se describe a continuacion, mediante el diagrama.



4 Estructura del aplicativo (Solucion).

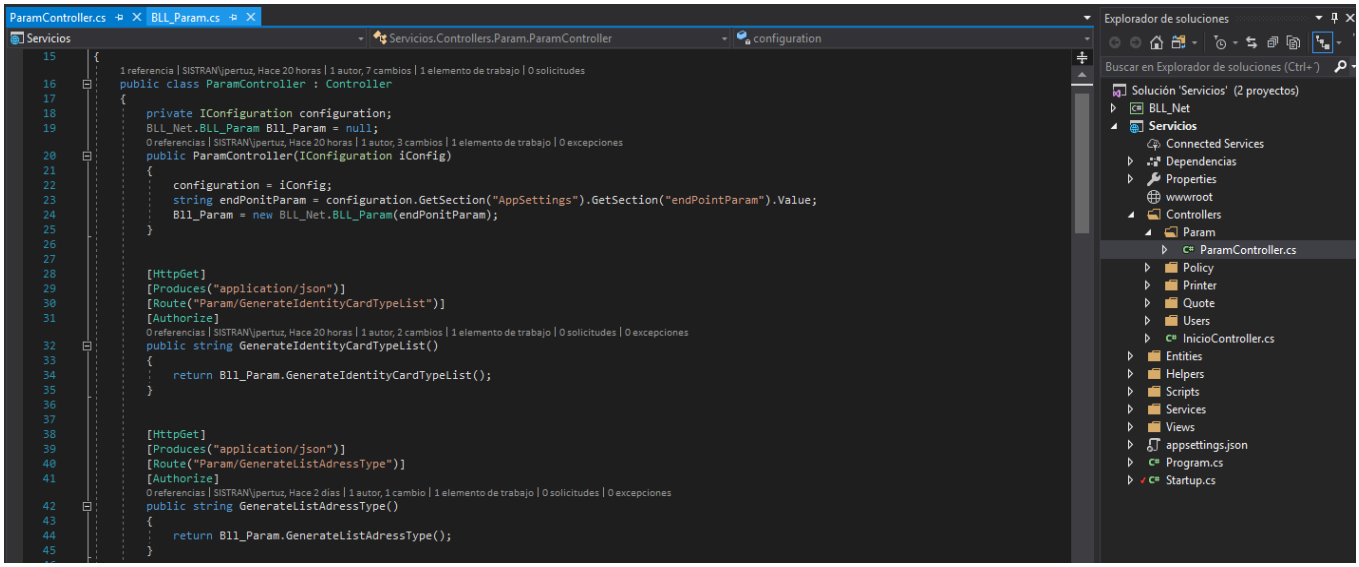
La WebApi realizada tiene la funcion de exponer una serie de servicios como tipo RestFull, la URL y/o servidor dependen de la configuracion realizada por el area de despliegue.
La WebApi, consta de 4 servicios de tipo RestFul, creados utilizando .Net Core 2.0, cada uno con sus respectivos métodos, lo cuales son:

- **Token:** Permite validar la autentcacion d elas peticiones al servidor.
- **Param:** Permite consultar los metodos de parametrizacion de la aplicación.
- **Quote:** Permite consultar los metodos de cotizacion de una poliza.
- **Printer:** Permite realizar el proceso de impresion de una poliza.
- **Policy:** Permite realizar el proceso de emision de una poliza.

Nota: Para consumir cualquiera de los metodos expuestos se debe realizar la previa autenticacion con usuario y contraseña, con el fin de obtener el token de acceso, como se describe mas adelante.

La solucion se encentra ordenda de la siguiente manera:

En el proyecto principal (Servicios), se encuentran los controladores, uno por cada servicio expuesto y en los cuales internamente se encuentran los diferentes metodos.



Cada metodo contiene la invocacion a su respectiva librería BLL, en este caso tomremos como ejemplo la de BLL_Param, la cual es la encargada de crear el cliente para el consumo de los servicios Soap expuestos por AXA.

Como parametros de entrada a las librerias (constructores) BLL, se tienen los diferentes endpoints donde se encuentrn expuestos los servicios Soap, los cuales son leidos de appsettings.json, en el proyecto principal y que adicionlmente contiene los vlores para el secreto (jwt) y la cadena de conexión a la base de datos.

Para los diferentes metodos en la librería, se realiza el llamado correspondiente, previamente deserializando mediante **JsonConvert.DeserializeObject** los parametros de entrada, los

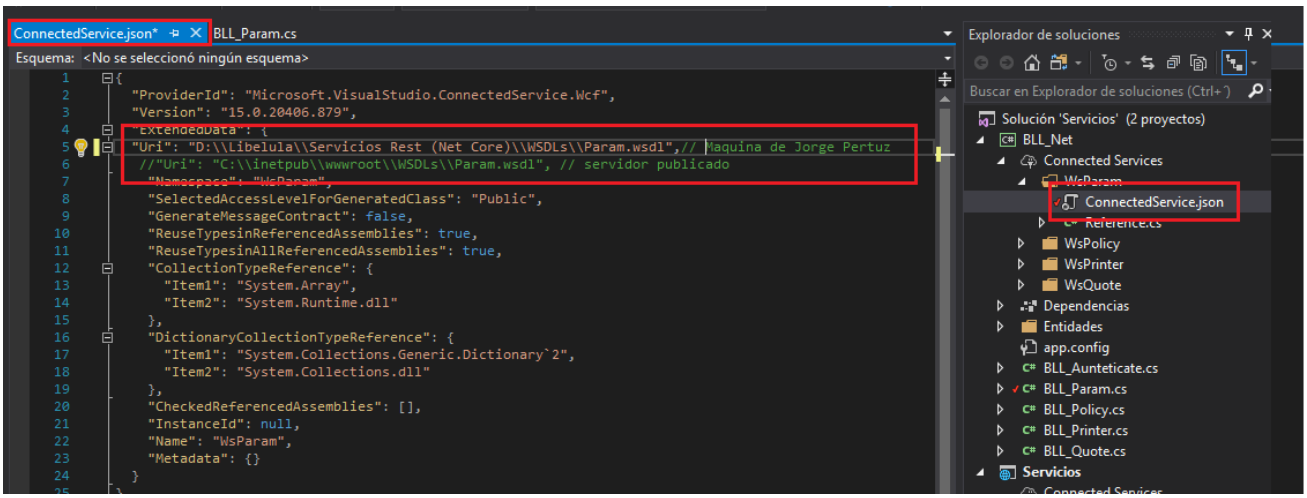
cuales son de tipo **JObject**, posteriormente se consume el sercivio Soap de forma asincrona y se retorna la respuesta, en caso de obtener error, ocurre una excepcion.

```
namespace BLL_Net
{
    3 referencias | SISTRAN\jpertuz, Hace 20 horas | 1 autor, 6 cambios | 1 elemento de trabajo
    public class BLL_Param
    {
        private WsParam.ExternalParamServiceClient Cliente = null;
        public BLL_Param(string endPointparam)
        {
            Cliente = new WsParam.ExternalParamServiceClient(
                      WsParam.ExternalParamServiceClient.EndpointConfiguration.BasicHttpBinding_IExternalParamService1,
                      endPointparam);
        }

        1 referencia | SISTRAN\jpertuz, Hace 20 horas | 1 autor, 5 cambios | 1 elemento de trabajo | 0 excepciones
        public string GenerateIdentityCardTypeList()
        {
            try
            {// try por si la deserializacion del objeto falla con relacion al tipo de dato request
                System.Threading.Tasks.Task<WsParam.GenerateIdentityCardTypeListResponse> Response = Cliente.GenerateIdentityCardTypeListAsync();
                Response.Wait();
                return JsonConvert.SerializeObject(Response.Result.Body.GenerateIdentityCardTypeListResult);
            }
            catch (Exception e)
            {
                return JsonConvert.SerializeObject("Error Consumiendo Servicio");
            }
        }
```

Un archivo importnte dentro del proyecto principal es StartUp, el cual contiene toda la configuracion del componente JWT.

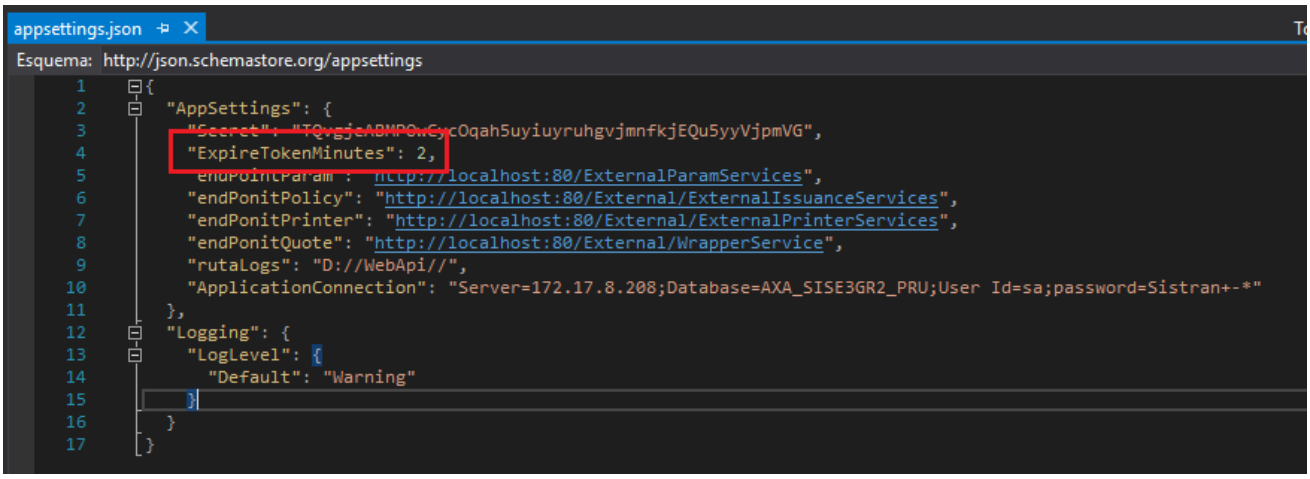
```
// This method gets called by the runtime. Use this method to add services to the container.
0 referencias | SISTRAN\jpertuz, Hace 4 días | 1 autor, 5 cambios | 1 elemento de trabajo | 0 excepciones
public void ConfigureServices(IServiceCollection services)
{

    // configure strongly typed settings objects
    var appSettingsSection = Configuration.GetSection("AppSettings");
    services.Configure<AppSettings>(appSettingsSection);

    // configure jwt authentication
    var appSettings = appSettingsSection.Get<AppSettings>();
    var key = Encoding.ASCII.GetBytes(appSettings.Secret);
    var conection = Encoding.ASCII.GetBytes(appSettings.ApplicationConnection);


    services.AddAuthentication(x =>
    {
        x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(x =>
    {
        x.RequireHttpsMetadata = false;
        x.SaveToken = true;
        x.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(key),
            ValidateIssuer = false,
            ValidateAudience = false,
            ValidateLifetime = true,
            RequireExpirationTime = true,
            ClockSkew = TimeSpan.FromMilliseconds(2)
        };
    });
```

Para la configuracion d elos servicios Soap, se tienen los archivos WSDL, los cuales se encuentran alojados en una ruta fisica de la mawquina y puede ser modificada mediente el archivo **ConnectedService.json** de cada servicio.



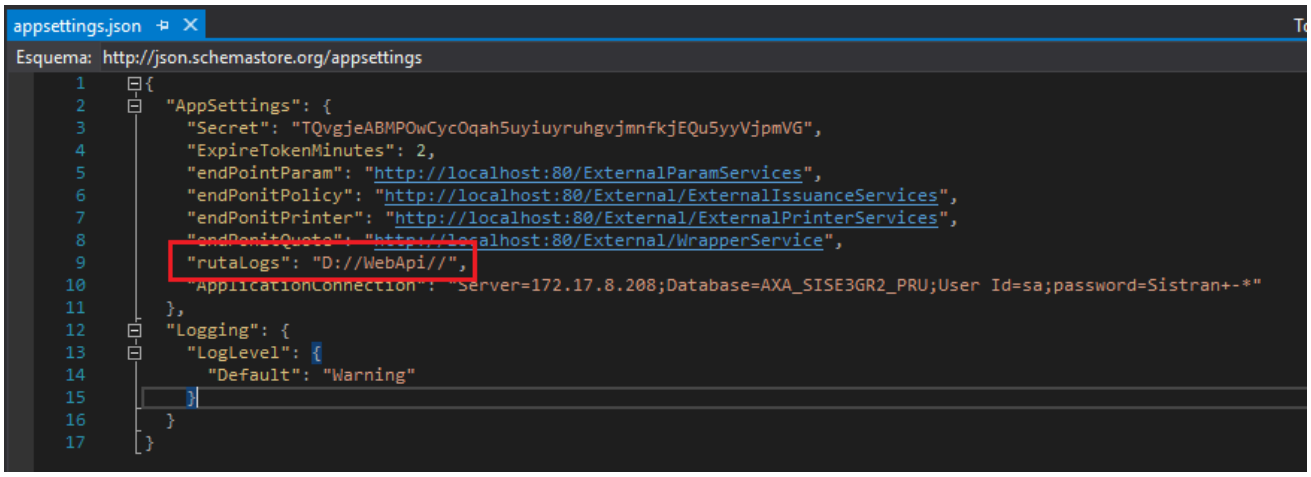
Para la utilizacion del TokenRefresh para generar autenticacion sin ingresar usuario y password, se tiene un GUID, el cual se entrega con el Token principal que cambia cada vez que se solicite un nuevo token realizando su respectiva validacion contra la base de datos.

El tiempo de expiracion del token es parametrizable y se encuentra establecido en minutos a traves de la variable `ExpireTokenMinutes` ubicado en el archivo `appsettings.json`



5 Captura de excepciones.

Como medida de seguimiento, se implemento el modulo de Logs para la aplicación, el cual simplemente captura las excepciones generadas en caso de algun error y de esta manera poder identificar y solucionar el problema.
Para su configuracion, se tiene una variable que contiene la ruta en la cual se pretende alojar el archivo `LOGS_WebApi.txt`, llamada `rutaLogs` ubicada en el archivo `appsettings.json` del proyecto principal.



6 Configuracion base de datos.

Para realizar la implementacion de autenticacion de usuarios/roles y JWT, se implemento una nueva tabla en la base de datos que es la encargada de manejar las sesiones y alojar los token´s para las autentoicaciones.

```
-- creacion d ela tabla para auenteicacion de usuarios
IF EXISTS(		SELECT 1
				FROM SYS.objects O
				WHERE O.NAME='USERS_INTEGRATIONS_WEB_API'
			)
		BEGIN
			DROP TABLE UU.USERS_INTEGRATIONS_WEB_API
		END
GO
CREATE TABLE UU.USERS_INTEGRATIONS_WEB_API(
ID int  NOT NULL IDENTITY(1,1),
FIRST_NAME_USER VARCHAR (255),
LAST_NAME_USER VARCHAR(60),
NAME_USER VARCHAR(60),
PASSWORD_USER VARCHAR(60),
ROLE_USER VARCHAR(60),
TOKEN_USER VARCHAR(255),
TOKEN_REFRESH_USER VARCHAR(255),
primary key(ID)
)
GO

-- creacion de procedimiento almacenado para listar los usuarios parametrizados
IF EXISTS(		SELECT 1
				FROM SYS.objects O
				WHERE O.NAME='SP_LIST_USERS_INTEGRATION_WEB_API'
			)
		BEGIN
			DROP PROCEDURE SP_LIST_USERS_INTEGRATION_WEB_API
		END
GO
CREATE PROCEDURE SP_LIST_USERS_INTEGRATION_WEB_API
AS
/*
AUTOR: Jorge David Pertuz Egea
FECHA: 09/Mayo/2019
MOTIVO: Prueba concepto servicios rest a partir de Soap [autenticacion de usuarios WebApi]
*/
BEGIN
		SELECT  ID,
				FIRST_NAME_USER,
				LAST_NAME_USER,
```

```
                                NAME_USER,
                                PASSWORD_USER,
                                ROLE_USER,
                                TOKEN_USER,
                                TOKEN_REFRESH_USER
        FROM UU.USERS_INTEGRATIONS_WEB_API
END
GO


 -- creacion de procedimiento almacenado para actualizar el token y tokenRefresh de los usuarios parametrizados
IF EXISTS(      SELECT 1
                        FROM SYS.objects O
                        WHERE O.NAME='SP_UPDATE_USERS_INTEGRATION_WEB_API_TOKEN'
                 )
        BEGIN
                DROP PROCEDURE SP_UPDATE_USERS_INTEGRATION_WEB_API_TOKEN
        END
GO
CREATE PROCEDURE SP_UPDATE_USERS_INTEGRATION_WEB_API_TOKEN
(
@PI_ID INT,
@PI_TOKEN_USER VARCHAR(255),
@PI_TOKEN_REFRESH_USER VARCHAR(255)
)
AS
/*
AUTOR: Jorge David Pertuz Egea
FECHA: 09/Mayo/2019
MOTIVO: Prueba concepto servicios rest a partir de Soap [autenticacion de usuarios WebApi]
*/
BEGIN
UPDATE UU.USERS_INTEGRATIONS_WEB_API
SET TOKEN_USER=@PI_TOKEN_USER,
        TOKEN_REFRESH_USER=@PI_TOKEN_REFRESH_USER
WHERE ID=@PI_ID
SELECT  ID,
                                FIRST_NAME_USER,
                                LAST_NAME_USER,
                                NAME_USER,
                                PASSWORD_USER,
                                ROLE_USER,
                                TOKEN_USER,
                                TOKEN_REFRESH_USER
        FROM UU.USERS_INTEGRATIONS_WEB_API
        WHERE ID=@PI_ID
END
```

Nota: Los fuentes de la WebApi, se encuentran disponibles en el repositorio Git en la siguiente url: http://arbutfs02:8080/tfs/Colombia/INTEGRACIONES/_git/WebServices_Libelula