Componente Java AFIP

Versión 1.1.0

Documentación

Table of Contents

Inicio rápido

Hay 2 formas de probar rápido cómo funciona el componente:

Ejecutar las pruebas unitarias

Hay 3 subproyectos:

- ar.com.system.afip
- ar.com.system.afip.android
- ar.com.system.afip.jre

En las pruebas unitarias del proyecto ar.com.system.afip se encuentra una interfaz Constants package ar.com.system.afip.util;

```
public interface Constants {
    String COMPANY = "<colocar el nombre de la empresa>";
    String UNIT = "<colocar el nombre de la unidad>";
    String CUIT = "<colocar el CUIT>";
    String ADDRESS = "<colocar la direccion>";
    String LOCATION = "<colocar ciudad y provincia>";
    String PRIVATE_KEY = "<colocar la clave privada>";
    String PUBLIC_KEY = "<colocar la clave publica>";
    String CERTIFICATE = "<colocar el certificado>";
    String CN = "<colocar el CN>";
}
```

donde se deben especificar los distintos parámetros para poder ejecutar los tests unitarios.

El requerimiento de firma de certificado (CSR - certificate signing request) se puede generar usando, por ejemplo, OpenSSL - AFIP provee hay una explicación de como hacerlo, pero una de las ventajas de usar esta biblioteca es que la generación puede hacerse desde código Java (y por lo tanto incluirse dentro de la misma aplicación). Por ejemplo, se podría escribir una clase como la siquiente (en las pruebas unitarias del proyecto ar.com.system.afip.jre):

```
import ar.com.system.afip.wsaa.business.api.Service;
import ar.com.system.afip.wsaa.business.api.WsaaManager;
import ar.com.system.afip.wsaa.data.api.CompanyInfo;
import ar.com.system.afip.wsaa.data.api.TaxCategory;
import ar.com.system.afip.wsaa.data.api.WsaaDao;
import org.junit.Test;

import java.util.Date;

import static ar.com.system.afip.utils.Utils.initWsaa;
import static ar.com.system.afip.wsaa.WsaaComponents.wsaaDao;
import static ar.com.system.afip.wsaa.WsaaComponents.wsaaManager;

public class EjemploCsrAfip {
    @Test
    public void generarCsr() {
        initWsaa();

        WsaaDao wsaaDao = wsaaDao();
```

```
        wsaaDao.saveCompanyInfo(new CompanyInfo(0, // El ID, usar 0 porque es el
valor que se usa en las pruebas unitarias
                "Austral Construcciones", // Nombre de la empresa
                true, // Activa?
                "rutas", // La unidad (area?)
                "30708372729", // el CUIT
                null, // La clave publica, valor nulo por ahora
                null, // La clave privada, valor nulo por ahora
                null, // El certificado, valor nulo por ahora
                "", // Ingresos brutos, opcional, no se usa
                new Date(), // Fecha de inicio de actividades, opcional, no se
usa
                TaxCategory.MONOTRIBUTO, // Condicion frente al IVA, opcional,
no se usa
                null, // Direción, opcional, no se usa
                null, // Ciudad, opcional, no se usa
                "austral" // alias - usado en el campo "Nombre simbólico del DN"
en AFIP
        ));

        WsaaManager wsaaManager = wsaaManager(Service.WSFE); // podría ser
Service.WSMTXCA, para el caso es lo mismo

        // Generación de clave pública y privada
        wsaaManager.initializeKeys();

        CompanyInfo companyInfo = wsaaDao.loadActiveCompanyInfo();

        System.out.println("Clave pública:");
        System.out.println(companyInfo.getPublicKey());
        System.out.println("Clave privada:");
        System.out.println(companyInfo.getPrivateKey());

        // Generación del CSR
        String csr = wsaaManager.buildCertificateRequest();
        System.out.println("CSR:");
        System.out.println(csr);
    }
}
```

Al ejecutarse, los resultados se mostrarán por consola. Vale notar que las claves son generadas cada vez, de modo que deberán guardarse las mismas a fin de usarlas posteriormente (de hecho la clave pública no es necesaria, la información se extrae de la clave privada).

Una vez que se tiene el CSR, se debe crear un certificado en AFIP (normalmente esto se hace en el ambiente de homologación primero, es decir, el ambiente de pruebas - podría salir muy caro probar en el ambiente de producción).

Para una información detallada sobre cómo funcionan los servicios de AFIP, conviene ir a la documentación. Pero para simplificar, basta con ingresar al sitio de AFIP con clave fiscal e instalar la aplicación WSASS - Autogestión Certificados Homologación. Es bastante intuitiva a pesar de todo. Cabe recordar que hay que asociar todos los servicios a usar (WSFE, MTXCA, etc.) al certificado en cuestión.

Una vez generado el certificado, el mismo se puede colocar en la clase Constants junto con el resto de los datos y ejecutar las pruebas unitarias. Vale notar que las mismas deben ejecutarse desde el

proyecto correspondiente a la plataforma a utilizar (ar.com.system.afip.android o ar.com.system.afip.jre).

Ejecutar la aplicación demo para Android

La aplicación de ejemplo se encuentra en el siguiente proyecto:

https://github.com/NibiruOS/afip-android-demo

Los pasos a seguir para la generación del certificado son los mismos que los explicados en el paso anterior, salvo que los datos deberán ser cargados en la pantalla de configuración en lugar de hacerlo en una clase de pruebas unitarias.

Arquitectura

Inyección de dependencias

En términos generales, las clases están pensadas para ser usadas utilizando el patrón de inyección de dependencias.

Bibliotecas para inyeccción de dependencias

Existe una especificación estándar de Java (JSR 330) que define anotaciones comunes. Todas las clases están configuradas con anotaciones estándar de JSR 330, por lo que se puede ysar cualquier inyector de dependencias que las soporte. Por ejemplo:

- Dagger 2
- Guice
- Spring Framework

Separación entre interfaz e implementación

Para implementar el patrón de inyección de dependencias de manera apropiada, se debe realizar una separación entre las interfaces (contratos) y sus correspondientes implementaciones (clases concretas). Por convención, este proyecto incluye las interfaces en subpaquetes de un paquete denominado api y las implementaciones bajo un paquete denominado impl.

Subproyectos

El proyecto está organizado en 3 subproyectos:

- ar.com.system.afip: Contiene los componentes comunes.
- ar.com.system.afip.android: Contiene componentes especificos para correr en plataforma android.
- ar.com.system.afip.jre: Contiene componentes para ejecutar en un ambiente Java estándar.

Servicios de afip

A su vez, dentro de cada proyecto, los paquetes están organizados de acuerdo al servicio de AFIP con el cual interactúen. Por ejemplo, en los componentes comunes:

- ar.com.system.afip.wsaa: Servicio WSAA.
- ar.com.system.afip.wsfe: Servicio WSFE.
- ar.com.system.afip.wsmtxca: Servicio MTXCA.

Agrupación por capa

Dentro de cada módulo para cada servicio de AFIP, los subpaquetes están organizados por capa. La convención de nombres es la siguiente:

- data: Persistencia de datos local (DAOs, modelo de dominio, etc).
- bussines: Lógica de negocios.
- service: Acceso a servicios remotos de AFIP.

Las capas de datos y servicios tienen implementaciones diferentes de acuerdo a la plataforma (Android o JRE), mientras que (en términos generales, pero no siempre) la capa de negocios es genérica.

Componentes

Módulo WSAA

Capa de datos

- WsaaDao: DAO para leer y guardar los datos de la empresa (clase CompanyInfo). Esta clase contiene todos los datos de la empresa requeridos por AFIP, más algunos que sirven solo para uso general. A fin de brindar la posiblidad de que la aplicación controle más de una empresa a la vez, hay un campo que determina si la empresa es la que está activa.
- SetupDao: DAO que permite leer la configuración (clase Setup). Se proveen 2 implementaciones, para los ambientes de homologación y de producción. Vale notar que se asume que sólo un servicio de facturación se utilizará a la vez (WSFE o MTCXA), por lo que la clase Setup sólo provee una dirección de WSDL de servicio de facturación.
- CredentialsDao: Permite leer y guardar las credenciales (clase Credentials). De esta forma, mientras las credenciales sean válidas, no será necesario realizar un nuevo login contra AFIP.

Datos de la compañía

Como se dijo antes, la clase CompanyInfo contiene los datos de la empresa. A continuación se detallan los mismos:

Datos de uso interno:

- id: Un ID interno, utilizado por el mecanismo de persistencia utilizado. Por ejemplo, un entero incremental.
- active: Bandera que indica si la compañía es la activa. Sirve para poder tener datos de más de una empresa en la misma aplicación. Sólo debe haber una empresa activa a la vez.

Datos requeridos por AFIP:

- name: El nombre de la empresa.
- unit: La unidad (área).
- cuit: El CUIT.
- publicKey: La clave pública utilizada para validar las firmas digitales.
- privateKey: La clave privada utilizada para firmar digitalmente.
- certificate: El certificado provisto por AFIP.
- alias: El alias que se configuró en AFIP (en el DN).

Datos opcionales (no usados por AFIP):

- grossIncome: El número de ingresos brutos.
- activityStartDate: La fecha de inicio de actividades.
- taxCategory: La condición frente al IVA.
- address: La dirección.
- location: La ciudad, pueblo, etc. donde se encuentra la empresa.

Capa de servicios

- LoginCMS: Realiza un login contra AFIP. Este servicio envía y recibe un String, siguiendo los formatos especificados para Generación del documento del TRA (LoginTicketRequest.xml).

Capa de negocio

- WsaaManager: Su función principal es ejecutar la operación de login, pero además provee métodos para generar claves, CSR, actualizar el certificado.

- WsaaTemplate: Permite ejecutar un callback asegurando que se disponga de credenciales válidas para invocar los servicios de AFIP. Las credenciales son recuperadas/almacenadas utilizando una instancia de CredentialsDao. Si por algún motivo el servicio invocado lanzara una excepción de tipo CredentialsException, significa que las credenciales ya no son válidas. En este escenario, la implementación de WsaaTemplate realizará un nuevo login y ejecutará el callback de nuevo.
- XmlConverter: Convierte instancias de objetos a XML y viceversa. Se utiliza para generar el LoginTicketRequest.xml y procesar la respuesta. La implementación varía de acuerdo a la plataforma, por lo que fue necesario crear esta interfaz.

Módulo WSFE

Capa de servicios

- ServiceSoap: Cliente SOAP para el servicio WSFE, generado a partir de su WSDL.

Capa de negocio

- WsfeManager: Wrapper sobre ServiceSoap que tiene las siguientes responsabilidades:
  - Asegurar que haya credenciales válidas para invocar a los métodos de ServiceSoap (la implementación utiliza WsaaTemplate para este fin).
  - Realizar conversiones menores de datos.
  - Manejo de errores. Por ejemplo, analizar respuestas de error y lanzar excepciones. Donde aplique, lanzar una instancia de CredentialsException, de modo que WsaaTemplate la capture y realice un nuevo login.

Módulo MTXCA

Capa de servicios

- MTXCAServicePortType: Cliente SOAP para el servicio MTXCA, generado a partir de su WSDL.

Capa de negocio

- MtxcaManager: Wrapper sobre MTXCAServicePortType que tiene las siguientes responsabilidades:

- Asegurar que haya credenciales válidas para invocar a los métodos de MTXCAServicePortType (la implementación utiliza WsaaTemplate para este fin).
- Realizar conversiones menores de datos.
- Manejo de errores. Por ejemplo, analizar respuestas de error y lanzar excepciones. Donde aplique, lanzar una instancia de CredentialsException, de modo que WsaaTemplate la capture y realice un nuevo login.

Referencias

- http://www.afip.gob.ar/ws/
- http://www.afip.gov.ar/ws/WSAA/Especificacion_Tecnica_WSAA_1.2.0.pdf
- http://www.afip.gov.ar/ws/WSFE/WSFE-GuiaAdicionalParaElProgramador.pdf
- http://www.afip.gov.ar/fe/documentos/manualdesarrolladormtxv0.pdf
- https://www.openssl.org/
- http://www.afip.gov.ar/ws/WSAA/cert-req-howto.txt
- https://en.wikipedia.org/wiki/Dependency_injection
- https://www.jcp.org/en/jsr/detail?id=330
- https://google.github.io/dagger/
- https://github.com/google/guice
- https://spring.io/