UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería informática

TRABAJO FIN DE GRADO

Desarrollo de Conector para la consulta sobre el sistema de Sustitución de Certificados en Papel de las Administraciones Públicas

Iván Núñez Estacio
Tutor: Alberto Chacón Ludeña
Ponente: Francisco de Borja Rodríguez Ortiz

Julio 2019

Desarrollo de Conector para la consulta sobre el sistema de Sustitución de Certificados en Papel de las Administraciones Públicas

AUTOR: Iván Núñez Estacio
TUTOR: Alberto Chacón Ludeña

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2019

Resumen

El conector sobre el sistema de Sustitución de Certificados en Papel es una funcionalidad añadida a la aplicación de contratación pública de la empresa Pixelware que pretende facilitar la solicitud de información sobre licitadores a cualquier empresa pública que genere un expediente.

La aplicación ofrece varias opciones al cliente, donde cada una de ellas se encarga de obtener información distinta de un licitador en cuestión.

Cuando se recibe la información, la aplicación se encarga de parsearla y darle el formato necesario para que el usuario final pueda entender el resultado.

La implementación consta de una aplicación cliente que es la que maneja el usuario final, un servicio web que se comunica con el recubrimiento de SCSP (Sustitución de Certificados en Papel) para obtener la información y una API (Application Programming Interface) que maneja toda la lógica detrás de la aplicación.

Palabras clave

Conector, licitador, expediente, recubrimiento SCSP, API.

Abstract

The connector on the Sustitución de Certificados en Papel system is a feature added to the public procurement application of the Pixelware company that aims to facilitate the request for information on bidders to any public company that generates a file.

The application offers several options to the client, where each one of them obtains different information from the bidder.

When the information is received, the application parse it so the final client can read the result without any problem.

The implementation consists of a client application that is used by the user, a web service that communicates with the SCSP covering (Sustitución de Certificados en Papel) to obtain the information and the API (Application Programming Interface) that handles all the logic behind the application.

Keywords

Connector, bidder, file, SCSP covering, API.

Agradecimientos

Primero me gustaría dar las gracias a mi padre y a mi madre, ya que sin ellos no podría haber estudiado lo que quería, y no habría llegado hasta donde estoy. Me han apoyado siempre y me han facilitado todo lo que he necesitado durante mis estudios.

También quiero agradecer a mis amigos/as y compañeros que me han ayudado en todas las asignaturas que se me han ido atascando durante estos cuatro años.

Y por último a la empresa Pixelware por darme la oportunidad de realizar este proyecto con ellos, por la ayuda que me han ofrecido en todo momento y el ambiente de trabajo que me han proporcionado, el cual ha favorecido mucho que este trabajo de fin de grado haya sido una experiencia muy positiva.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1 Motivación

El desarrollo de esta aplicación viene de la necesidad de eliminar el papel de las administraciones públicas. Cada día se generan más y más expedientes de ofertas públicas, a los que se presentas muchas empresas. Tener todo esto en papel conlleva un gasto enorme de recursos, tanto de papel como de personal, ya que toda esa información hay que guardarla y mantenerla para cuando se requiera.

SCSP pretende eliminar el papel de las administraciones públicas y mantener todos los datos digitalizados, eliminando así el mal gasto de papel y además automatizando el servicio de información sobre una empresa.

Con esta aplicación se pueden consultar los datos de cualquier persona que haya presentado una oferta pública a algún expediente, ya sea desde datos sobre sus pagos con la TGSS o AEAT [2], hasta los títulos universitarios de alguno de los miembros de la empresa. Estas son solo unas pocas de las opciones que SCSP nos ofrece, aunque para el desarrollo de esta aplicación no fue necesario usar más por lo que el cliente especificó [1].

1.2 Objetivos

La integración de la plataforma de SCSP, desarrollado en la empresa Pixelware, tiene intención de facilitar a todas las empresas que presentan expedientes públicos para su adjudicación puedan obtener de una manera sencilla y rápida información sobre los licitadores que están interesados en su oferta.

Para el correcto desarrollo de la aplicación había que pasar por las fases de diseño y desarrollo, y luego realizar un conjunto de pruebas finales para asegurar el correcto funcionamiento de la aplicación antes de pasarlo al entorno del cliente.

Como parte del inicio del proyecto se establecieron unos objetivos concretos para saber si el proyecto iba por buen camino, y así asegurar la calidad del producto final.

Transparencia para el cliente: como toda aplicación que usará un usuario final, todo el proceso interno de la aplicación debe ser ajeno al cliente, es decir, este no tiene que desempeñar ninguna función más allá de “dar un click”, para así asegurar el correcto funcionamiento de la aplicación.

Tiempo de ejecución: los tiempos de carga de las páginas de la aplicación tenían que ser muy pequeños, los tiempos de ejecución en lado del servidor también tenían que ser muy reducidos y el tiempo de respuesta asequible. Este último no dependía tanto de nosotros ya que el servicio web al que invocamos puede funcionar de una forma distinta cada vez sin nosotros poder manejarlo.

Robustez de la aplicación: es obvio pensar que una aplicación que se basa en ofrecer información importante a una empresa que está decidiendo a quien concederle la oferta que han publicado no puede fallar constantemente y dar un mal servicio. Por eso, una de las prioridades era que la aplicación funcionase correctamente bajo cualquier circunstancia.

Integridad: la aplicación desarrollada no es independiente, ya que está integrada en la aplicación de contratación de Pixelware, por lo que tenía que integrarse con todo el resto de la aplicación sin problema, manteniendo los estilos de diseño y además facilitando la navegación hasta esta aplicación desde otras páginas.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Estado del arte:** descripción de las distintas tecnologías usadas en el proyecto además de una comparativa con las otras opciones que se manejaron al principio.
- **Diseño:** explicación de la arquitectura de la aplicación, del diagrama de clases y de la interfaz de usuario.
- **Desarrollo:** se explicará el desarrollo de la aplicación, la decisiones que se tomaron y cuáles no, y una breve ilustración del funcionamiento.
- **Integración, pruebas y resultados:** como se integra la aplicación con el resto de componentes, lista de pruebas que se llevaron a cabo junto con los resultados que se obtuvieron.
- **Conclusiones y trabajo futuro:** conclusiones del trabajo realizado, nombrando los problemas que se tuvieron a nivel personal y empresarial. Futuros desarrollos para esta aplicación.
- **Anexo I: Diagrama de mensajes de petición:** todos los diagramas que muestran la composición de los mensajes de petición a cada servicio implementado.

2 Estado del arte

2.1 Introducción

A día de hoy toda aplicación web está programa en Java o en .NET. Aunque .NET no sea un lenguaje como tal, sino que sea el framework de Microsoft, entendemos por .NET la combinación del lenguaje C# y ASP.NET, ya que son los usados en este desarrollo aunque existen otros muchos como Visual Basic C++, F# etc.

La aplicación que hemos desarrollado está en .NET totalmente por varias razones, la primera y más directa es que la aplicación de Pixelware está en .Net y había que seguir esa línea. La segunda razón es que .NET es un conjunto de elementos que funcionan muy bien entre ellos, y esto Java no lo tiene, y al final puede llevar a errores de compatibilidad entre versiones.

A continuación hablo un poco de las opciones que se tenían en un principio y cuales hemos usado:

2.2 Frontend

2.2.1 Javascript y PHP

Javascript[1] es un lenguaje del lado del cliente que no puede faltar en ninguna aplicación web. Es imprescindible ya que facilita muchísimo el trabajo a la hora de comunicar cliente con servidor.

Por otro lado tenemos PHP[2], que puede usarse para lo mismo, pero este lenguaje no es tan versátil como JS y además tiene un problema respecto a JS, y es que éste último es muy parecido a Java y no requiere de mucho aprendizaje para dominarlo. En cambio, PHP es un lenguaje independiente, por lo que tiene su propia sintaxis y funciones, y hace el desarrollo más tedioso. Además, JS te permite el uso de Jquery[3], del cual hablaremos más adelante.

2.2.2 HTML y ASP.NET

Pasando a la parte de diseño, HTML[4] es el lenguaje más común para el desarrollo de páginas web. Es claro, sencillo y junto con JS y JQuery puedes hacer de todo. Pero en nuestro caso, al haber decidido usar .NET, ya teníamos un lenguaje para el diseño de páginas, el propio de Microsoft.

ASP.NET[5] es básicamente lo mismo que HTML además de añadidos hechos por Microsoft para que su framework funcione por detrás, es decir, por el lado del servidor. ASP.NET, al igual que HTML, puede usar JS y JQuery, pero además tiene un extra que facilita mucho el control Cliente-Servidor, y es el atributo “runat=server” que se le puede añadir a cualquier

[1] https://es.wikipedia.org/wiki/JavaScript

[2] http://php.net/manual/es/intro-whatis.php

[3] https://api.jquery.com/

[4] https://es.wikipedia.org/wiki/HTML

[5] https://es.wikipedia.org/wiki/ASP.NET

etiqueta HTML. Este atributo lo que nos dice es que todo lo que esté dentro de esa etiqueta se puede leer, modificar… desde el lado del servidor, permitiendo así mayor transparencia al cliente.

2.2.3 JQuery

Jquery[6] no es un lenguaje nuevo, es una biblioteca propia de JS que permite una interacción con los elementos HTML mucho más clara que el propio JS. Además permite crear eventos y manipularlos como necesitemos. En este apartado Jquery es claramente la opción más viable, podríamos usar únicamente JS pero sería desperdiciar el potencial de Jquery.

2.3 Backend

2.3.1 Java

Pasando al lado del servidor, hablaremos primero de Java[7]. Es un lenguaje orientado a objetos, que es muy bueno para aplicaciones web. Es software libre y funciona en cualquier SO. Para nuestro caso no sería una mala elección ya que no tenemos que hacer una aplicación de escritorio, donde Java es más engorroso ya que su programación visual no es muy trivial, pero al tratarse de una aplicación web. Java funciona perfectamente.

2.3.2 .NET

Por otro lado, tenemos .NET[8], el framework de Microsoft. Dentro de .NET tenemos la opción de usar C#, que es un lenguaje imperativo orientado a objetos, prácticamente igual a Java ya que Microsoft se basó en él para crear C#, aunque tiene ligeras variaciones con respecto a Java, como puede ser la programación funcional que se ha añadido en las últimas versiones. Pero para la programación funcional ya está F#, aunque no hablaremos de él ya que no la usamos.

Sin embargo, al contrario de Java, .NET no es software libre y solo funciona en sistemas operativos Windows, aunque hay iniciativas como Mono[9] que intentar hacer .NET compatible con otros SO.

2.4 Conclusión

Como he comentado al principio, acabamos por elegir .NET sobre Java ya que la aplicación de Pixelware está completamente en .NET. A parte de eso, hemos usado JS para la comunicación entre el cliente y el servicio web desarrollado en el Servidor, junto con JQuery para manipular el DOM de las páginas de ASP.NET.

---

[6] https://es.wikipedia.org/wiki/JQuery

[7] https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)

[8] https://www.microsoft.com/net/learn/what-is-dotnet

[9] http://www.mono-project.com/

3 Diseño

3.1 Requisitos funcionales

Como todo proyecto que involucra a un cliente hay que empezar por sacar los requisitos funcionales que quiere que tenga la aplicación. En nuestro caso, al ser un evolutivo sobre una aplicación, no teníamos muchos requisitos funcionales, y no entrevistamos directamente al cliente, ya que por funcionamiento de la empresa esa tarea la realiza un consultor. A nosotros nos llegó el documento [2], que es la característica del proyecto, donde se explica que se quiere, como se quiere que se vea y que requisitos funcionales y no funcionales son necesarios. A continuación voy a ir nombrando los distintos RF.

3.1.1 RF1 Servicios a implementar de SCSP

El primer requisito que se nos impuso era la necesidad de implementar tres servicios de SCSP en concreto. Esto no supuso ningún problema, porque todos los servicios funcionan de la misma manera y desde el mismo servidor de SCSP aunque si influyó a la hora de especificar las clases que íbamos a usar, pero esto lo comentaré más adelante.

3.1.2 RF2 Llamada al TramitadorWeb

Primero voy a explicar que es el TramitadorWeb ya que es la primera vez que aparece. El TramitadorWeb es la aplicación de Pixelware que trata todos los temas de contratación. Esta aplicación solo es accesible a través de unos formularios creados a base de XML y personalizaciones. Básicamente, lo que este RF quiere decir, es que nuestro desarrollo debía ser accesible a través de una URL específica que invocase al TramitadorWeb. Esta URL se oculta al cliente porque usa parámetros por el método GET.

3.1.3 RF3 Parámetros URL

Lo separo del anterior porque al final son dos cosas distintas, pero están relacionadas. Se especificó que la llamada a nuestro desarrollo se realizase a través de una URL como ya he dicho, pero que el único parámetro fuese el número de expediente en el que se estaba accediendo a la información. Con ese número de expediente teníamos que ser capaces de sacar el resto de información necesaria para el desarrollo.

3.1.4 RF4 Lista licitadores del expediente

En un expediente público se pueden presentar varios licitadores, cada uno con su propuesta, por eso se pedía que al pedir información primero se mostrase una lista de todos los licitadores que habían ofertado en ese expediente. Esta lista estaría dividida por los lotes que se ofertaban dentro del expediente. Esta lista lo que nos permitía era diferenciar de quién queríamos pedir información, para en un futuro mandar un identificador a SCSP dependiendo de qué licitador estábamos consultando.

3.1.5 RF5 Mostrar todos los informes disponibles

Este requisito tiene en cuenta los desarrollos futuros que se llevarán a cabo, visitar el punto 6.2 para saber más acerca de esto.

Desde un principio se quiso que la página principal del proyecto no fuese solo para SCSP, si no que diese lugar a otros tipos de servicios que ofreciese información sobre licitadores. Esto se quiso así porque uno de los objetivos principales de esta parte de la aplicación es darle al cliente información necesaria para poder elegir a que licitador darle la propuesta publicada y no era viable crear una página distinta para cada uno de los servicios que se iban a desarrollar.

3.1.6 RF6 Abrir pestaña nueva para los resultados

Como último RF se estableció que una vez obtenida la información que el cliente desease, se mostrase el resultado en una ventana nueva, dejando así la posibilidad de pedir información distinta y tenerla toda abierta para facilitar su uso.

3.2 Requisitos no funcionales

No me voy a extender mucho en este apartado porque los requisitos no funcionales son muy básicos y son los que toda aplicación web requiere.

3.2.1 RNF1 Velocidad de carga de páginas

Al tratarse de una aplicación web, la velocidad de navegación entre las distintas páginas web debe ser rápida y fluida.

3.2.2 RNF2 Velocidad al solicitar la información

Este requisito tiene que ver con el primero, pero al tratarse de una comunicación con un servidor ajeno pueden darse muchas situaciones. La idea principal es que el tiempo de espera al solicitar información de un licitador sea razonable. Claramente esto puede verse afectado por la carga de la red cuando se haga la solicitud, pero desde nuestro lado al menos tenemos que asegurar que se haga lo más rápido posible.

3.2.3 Seguridad de la aplicación

Aunque nuestro desarrollo esté dentro de una aplicación la cual ya tiene sus medidas de seguridad, también debemos asegurar que desde nuestro proyecto no haya fallas de seguridad que pongan en peligro al cliente.

3.2.4 Usabilidad de la aplicación

Como toda aplicación web, su usabilidad debe ser alta, permitiendo que el usuario final pueda obtener todo lo que desee sin tener que realizar tareas muy costosas.

3.3 Diseño de las páginas web

Algo importante de una aplicación web es la claridad con la que se muestran los elementos al usuario final. Por ello tuvimos un periodo de diseño en el que decidimos como queríamos montar la página web para que fuese lo más sencillo posible de navegar.

Partimos de la lista de los licitadores que era muy sencillo de hacer, pues era una tabla con los distintos licitadores y un botón que permitiese obtener la información de cada uno. Desde esa página debíamos acceder a las distintas formas de solicitar información de un licitador.

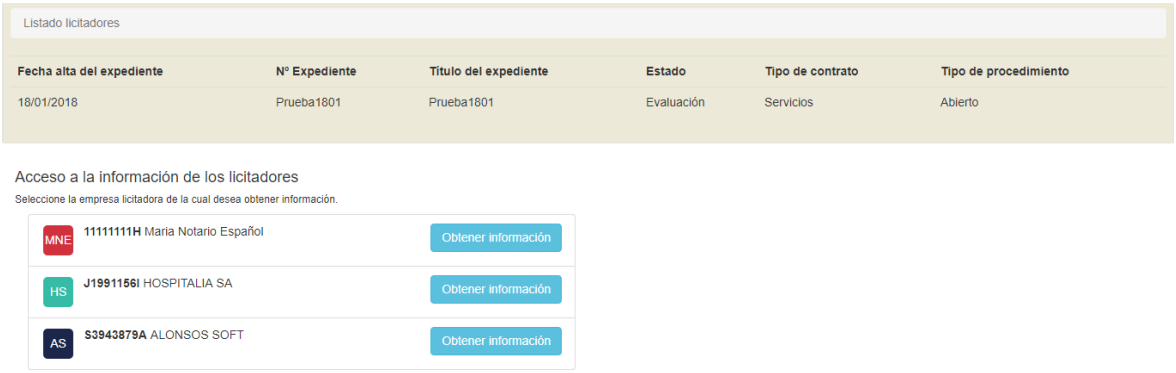

Figura 1 - Listado de los licitadores de una publicación

En la página de los informes tuvimos que pararnos más tiempo, porque iba a llevar mucha carga en un futuro. Decidimos hacer un bloque por servicio, y dentro de cada bloque mostrar todas las opciones posibles para ese servicio.

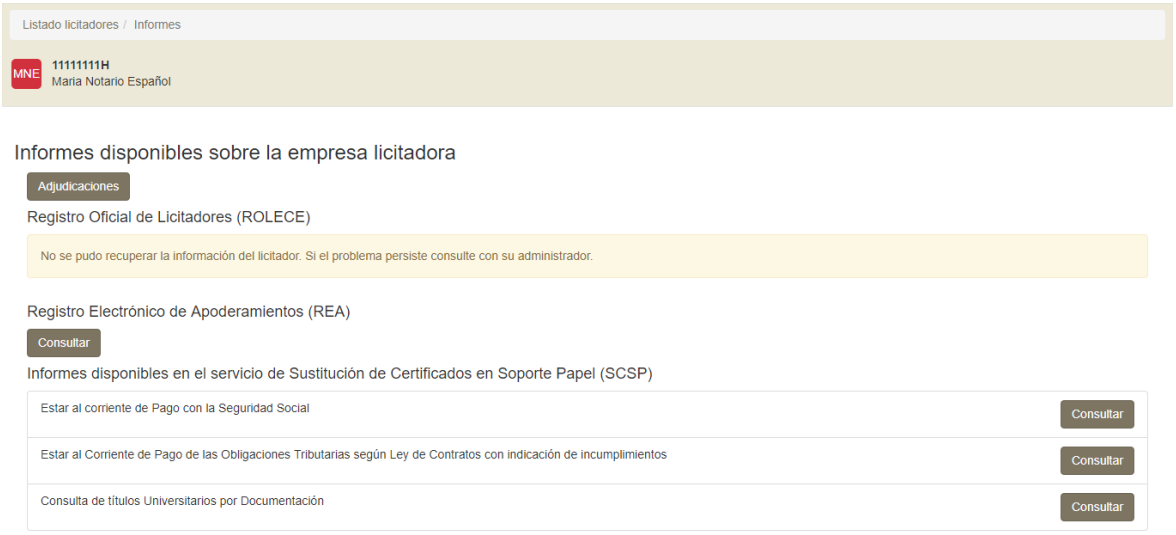

Figura 2 - Listado de servicios para solicitar información

Como se observa en la figura 2, hay tres bloques significativos, ROLECE, REA y SCSP. De estos tres servicios solo estaba implementado el tercero, pero decidimos ponerlos todos para tener la estructura clara, aunque no hiciesen nada. El primer botón es la funcionalidad propia de Pixelware que comento en el apartado 6.2, el cual muestra las adjudicaciones de la empresa licitadora.

Hablando de la estructura general de las páginas, en la parte superior se muestra información útil respecto al expediente (en la figura 1), y el nombre y el documento del licitador (figura 1 y figura 2). Esto se quiso así para que el cliente pudiese saber en todo momento donde se encontraba y sobre quien estaba pidiendo información.

En la figura 2, en el bloque de SCSP vemos los tres servicios que pidió el cliente. Simplemente se muestran y el cliente consulta el que quiere. Esto ayuda mucho a que la usabilidad de la página sea alta y que cualquier persona pueda hacerse cargo de esta solicitud de información. El único servicio que difiere del resto es el último, el de solicitar títulos universitarios por documentación. Esto es así porque analizando los tres servicios nos dimos cuenta de que los dos primeros eran lógicos para una empresa, pero el tercer solo tendría sentido para persona físicas, así que decidimos añadir una pequeña pantalla al pulsar el botón de consultar que pidiese el documento de identidad de la persona por la que queríamos solicitar los títulos (figura 4).

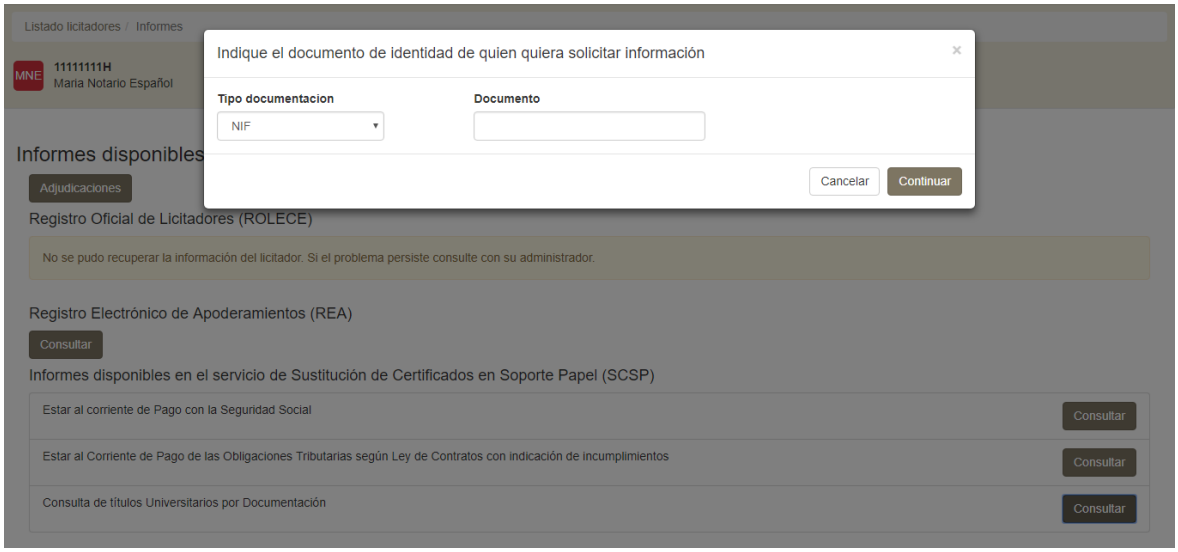

Figura 3 - Modal en solicitud de títulos universitarios por documentación

3.4 Diseño del Backend

Hemos estado hablando del diseño de la parte del cliente web, las páginas web, los requisitos de éstas etc. Ahora vamos a hablar sobre la parte del servidor, donde sí diseñé yo la estructura.

Primero voy a explicar un poco la estructura del TramitadorWeb[10]. Existen dos parte importantes, la parte cliente, que contiene todas las páginas web y la parte servidor, o como le llamamos nosotros ContratacionApi. Dentro de ContratacionApi reside toda la lógica de la aplicación.

[10] Aplicación principal de Pixelware

Lo principal fue diseñar la estructura que íbamos a usar para implementar el servicio de SCSP.

Dentro de ContratacionApi hay un módulo exclusivo de integraciones con servicios web, así que ahí metimos el nuestro. Dentro de la integración de SCSP tuvimos que dividir el código en tres partes:

1. La base de la integración. Esta base estaría compuesta de dos elementos, el principal encargado de crear la conexión, recibir los parámetros y pasárselos a la segunda parte, que se encargaría de crear la solicitud y devolverla.
2. Las clases que contendrían los datos recibidos desde SCSP.
3. Y por último una parte de funciones útiles que las usaremos para hacer el código más estructurado y que cada módulo solo trabaje lo que es suyo propio.

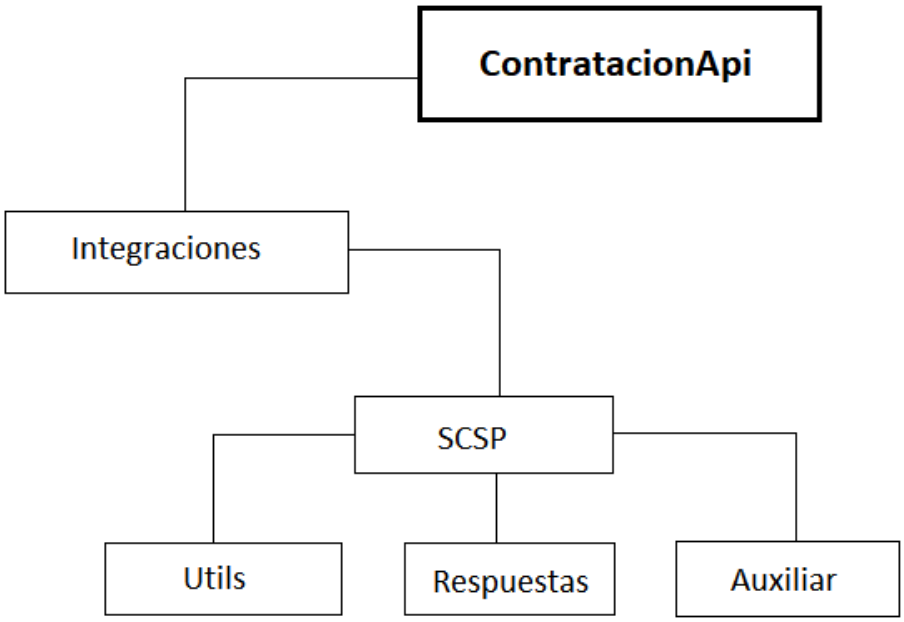

Figura 4 - Estructura carpeta integración SCSP

En la figura 4 se puede ver la distribución de carpetas que se tuvo en mente desde el principio, luego se irían añadiendo los distintos ficheros de código, pero de eso hablaré en el apartado de desarrollo.

4 Desarrollo

4.1 Frontend

4.1.1 Páginas web

Lo primero que desarrollamos fueron las páginas web que se mostrarían al cliente para así poder ir probando la comunicación cliente-servidor que teníamos que hacer.

Se crearon dos páginas web desde cero, aunque una de ellas está vacía pues se rellena al recibir la información de SCSP. Más adelante comentaré como rellenamos esta página y que HTML acaba teniendo.

La página principal de la aplicación es el listado de servicios disponibles (figura 2). No voy a mostrar todo el código del documento ya que hay partes que no pertenecen a este proyecto, por lo que solo pondré capturas de lo que desarrollé yo.

```
<div id="accionesSCSP" class="col-sm-12" style="display: none">
    <h4>
        <ml>Informes disponibles en el servicio de Sustitución de Certificados en Soporte Papel (SCSP)</ml>
    </h4>
    <div class="list-group">
        <span class="list-group-item clearfix">
            <span class="media-body">
                <ml>Estar al corriente de Pago con la Seguridad Social</ml>
            </span>
            <span class="media-right media-middle">
                <button class="btn btn-primary" type="button" value="Q2827003ATGSS001" onclick="return ObtenerHTMLInformeConsultaSCSP(this.value)">
                    <ml>Consultar</ml>
                </button>
            </span>
        </span>
        <span class="list-group-item clearfix">
            <span class="media-body">
                <ml>Estar al Corriente de Pago de las Obligaciones Tributarias según Ley de Contratos con indicación de incumplimientos</ml>
            </span>
            <span class="media-right media-middle">
                <button class="btn btn-primary" type="button" value="ECOT101I" onclick="return ObtenerHTMLInformeConsultaSCSP(this.value)">
                    <ml>Consultar</ml>
                </button>
            </span>
        </span>
        <span class="list-group-item clearfix">
            <span class="media-body">
                <ml>Consulta de títulos Universitarios por Documentación</ml>
            </span>
            <span class="media-right media-middle">
                <button class="btn btn-primary" type="button" value="SVDTUWS03" data-toggle="modal" data-target="#form-scsp" onclick="propagarCodigoSCSP(this.value)">
                    <ml>Consultar</ml>
                </button>
            </span>
        </span>
    </div>
</div>
```

Figura 5 - Parte HTML de los servicios de SCSP

Esta parte de código muestra las tres opciones que le damos al cliente para solicitar información a SCSP. Como ya comenté en el apartado de diseño, la tercera opción además muestra un modal en el que recogemos el documento de la persona por la que solicitamos los títulos universitarios.

```html
<div class="modal fade" id="form-scsp" role="dialog" style="display: none">
    <div class="modal-dialog modal-lg">
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal">&times;</button>
                <h4 class="modal-title">Indique el documento de identidad de quien quiera solicitar información</h4>
            </div>
            <div class="modal-body">
                <div class="row">
                    <div class="col-md-3">
                        <label for="tipo-documento" class="form-control-label">Tipo documentacion</label>
                        <select id="tipo-documento" class="custom-select form-control">
                            <option value="NIF">NIF</option>
                            <option value="NIE">NIE</option>
                        </select>
                    </div>
                    <div class="col-md-1"></div>
                    <div class="col-md-4">
                        <label for="documento" class="form-control-label">Documento</label>
                        <input type="text" class="form-control" id="documento" maxlength="9" />
                    </div>
                </div>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-default" data-dismiss="modal">Cancelar</button>
                <button id="boton-continuar-form" type="button" class="btn btn-primary" data-dismiss="modal" value="" onclick="return ObtenerHTMLInformeConsultaSCSP(this.value)">Continuar</button>
            </div>
        </div>
    </div>
</div>
```

Figura 6 - Modal para indicar el documento del solicitante

En un principio este modal está oculto, y solo se muestra al pulsar sobre el botón de consultar títulos universitarios.

La lógica de esta página es sencilla, cada botón contiene el código único de SCSP para identificar cada servicio.

```javascript
function ObtenerHTMLInformeConsultaSCSP(codigoSCSP) {
    MuestraPanelEspera();

    try {
        var args = {
            idUser: getRequestParam("iduser"),
            idSolicitante: $('#idSolicitante').val(),
            codigoSCSP: codigoSCSP,
            numExpediente: getRequestParam("numeroExpediente"),
            tipoDocumento: $('#tipo-documento').val(),
            documento: $('#documento').val(),
        }
        $.ajax({
            type: 'POST',
            contentType: 'application/json; charset=utf-8',
            url: document.URL.substr(0, document.URL.lastIndexOf('/')) + '/DataService.svc/ObtenerHTMLInformeConsultaSCSP',
            data: JSON.stringify(args),
            dataType: "json",
            async: false,
            error: function (jqXHR, textStatus, errorThrown) {
                if (typeof console != "undefined") console.error("ERROR LLAMADA ObtenerHTMLInformeConsultaSCSP", textStatus, errorThrown);
            },
            success: function (data) {
                if (data.d != null) {
                    var w = window.open("ResultadoSCSP.aspx");
                    w.onload = function () {
                        $(w.document.body).html(data.d);
                    };
                }
                else {
                    alert("<ml>No se pudo recuperar la información seleccionada. Si el problema persiste consulte con su administrador.</ml>");
                }
            }
        });
    } catch (err) {
        if (typeof console != "undefined") console.error("ERROR ObtenerHTMLInformeConsultaSCSP", err, err.description);
        OcultaPanelEspera();
    }

    OcultaPanelEspera();
    return false;
}

function propagarCodigoSCSP(codigoSCSP) {
    $('#boton-continuar-form').prop('value', codigoSCSP);
}
```

Figura 7 - Funciones JS para la integración de SCSP

Estas dos funciones hacen toda la lógica de cliente. La segunda “propagarCodigoSCSP” solo se usa en el tercer servicio, y sirve para pasar el código de ese servicio al siguiente botón que ya invocaría a la función “ObtenerHTMLInformeConsultaSCSP”. Esta función recibe el código de la opción que se ha pulsado y mediante JQuery realiza una llamada

Ajax[11] al método “ObtenerHTMLInformeConsultaSCSP” del servicio web implementado en el TramitadorWeb, DataService.svc. Este DataService.svc está compuesto de funciones que son invocadas mediante llamadas Ajax, y es el conector entre el cliente y el servidor, o como lo vemos nosotros, la conexión entre el TramitadorWeb y ContratacionApi.

```
[OperationContract]
[WebInvoke(Method = "POST",
    BodyStyle = WebMessageBodyStyle.WrappedRequest,
    ResponseFormat = WebMessageFormat.Json,
    RequestFormat = WebMessageFormat.Json
)]
public string ObtenerHTMLInformeConsultaSCSP(string idSolicitante, string codigoSCSP, string numExpediente,
    string tipoDocumento, string documento, string nombre, string apellido1, string apellido2)
{
    try
    {
        ModeloDatos modeloDatos = ModeloDatos.Clone();

        Expediente expediente = Expediente.ObtenerExpedientePorIdentificador(modeloDatos, numExpediente);

        Solicitante solicitante = Solicitante.ObtenerSolicitantePorId(idSolicitante);

        return PublicarSCSP.ObtenerInformeConsultaSCSP(expediente, solicitante, codigoSCSP, tipoDocumento, documento);
    }
    catch (Exception exception)
    {
        MyTraceSource.myTraceSource.TraceEvent(TraceEventType.Error, 0, "Error al obtener la información SCSP solicitada: '{0}'", exception);
        return null;
    }
}
```

Figura 8 - Definición del método ObtenerHTMLInformeConsultaSCSP del DataService.svc

Esto ya es código del servidor y no se ejecutará nada más en el lado del cliente hasta que esta función termine. Esta función es muy sencilla, simplemente obtiene el modelo de datos que estamos usando, el expediente en el cual estamos trabajando y el solicitante, que equivale a la persona que está logeada en la aplicación.

Esta información se saca de funciones propias de Pixelware y que no puedo mostrar por temas de privacidad. Pero a groso modo, el modelo de datos es un archivo XML que se deserializa, y el expediente y el solicitante se saca de la base de datos.

Una vez obtenidos estos parámetros, se llama a la base de la integración de SCSP que será explicado en la parte de Backend.

Antes de pasar con el Backend voy a explicar lo que hace el cliente web una vez ha terminado el servidor. El servidor devuelve un HTML puro, y si nos fijamos en la figura 7, en la función “ObtenerHTMLInformeConsultaSCSP” se ejecuta este código:

```
if (data.d != null) {
    var w = window.open("ResultadoSCSP.aspx");
    w.onload = function () {
        $(w.document.body).html(data.d);
    };
}
```

Figura 9 - Creación ventana nueva

[11] https://es.wikipedia.org/wiki/AJAX

La función window.open (string) crea una nueva ventana en el navegador que tendrá la estructura del archivo ASPX que le indiquemos (en este caso, podría ser una página HTML normal). Una vez creada esa ventana, le indicamos mediante Jquery que en el evento de "onload" cambie el contenido HTML del "body" por el de "data.d". Este "data.d" es el resultado devuelto por el DataService.svc, es decir, el HTML puro con los resultados de SCSP.

4.2 Backend

Una vez se ha invocado a la función del servicio web del TramitadorWeb, DataService.svc, este pasa el funcionamiento a ContratacionApi mediante la función estática "ObtenerInformeConsultaSCSP".

```
namespace ContratacionApi.Integraciones.SCSP
{
    public class PublicarSCSP
    {
        /// <summary>
        /// Obtiene la informacion solicitada mediante el codigo SCSP seleccionado sobre el licitador indicado.
        /// </summary>
        /// <param name="expediente">Expediente al que pertenece el licitador</param>
        /// <param name="solicitante">Datos de la persona que solicita informacion</param>
        /// <param name="codigoSCSP">Codigo SCSP que indica que información se quiere consultar</param>
        /// <param name="tipoDocumento">Tipo de documento del licitador (este parámetro solo se recibe cuando la consulta es de titulos universitarios)</param>
        /// <param name="documento">Documento del licitador (este parámetro solo se recibe cuando la consulta es de titulos universitarios)</param>
        /// <returns>HTML que contiene la informacion devuelta por el recubrimiento de SCSP</returns>
        public static string ObtenerInformeConsultaSCSP(Expediente expediente, Contratacion.Tramitacion.Solicitudes.Solicitante solicitante,
            string codigoSCSP, string tipoDocumento, string documento)
        {
            try
            {
                scspwsClient cliente = null;
                Respuesta respuesta = null;

                // Creamos la peticion sincrona, los datos especificos los dejo a null ya que ningún servicio que vamos a usar los necesita
                PeticionSincrona peticion = new PeticionSincrona()
                {
                    Atributos = Genericos.GetAtributos(codigoSCSP),
                    Solicitudes = Genericos.GetSolicitud(expediente, solicitante, codigoSCSP, tipoDocumento, documento)
                };

                cliente = new scspwsClient("scspwsSoap11");
                cliente.Open();

                MyTraceSource.myTraceSource.TraceEvent(TraceEventType.Verbose, 0, "Conectando al endpoint {0}", cliente.Endpoint.Address.ToString());

                respuesta = cliente.peticionSincrona(peticion);
                cliente.Close();

                return AuxiliarSCSP.ParseRespuestaSCSPToHtml(respuesta, codigoSCSP);
            }
            catch (Exception exception)
            {
                // Condicion para controlar los errores de SCSP que devuelve por excepcion, y no respuesta
                if(exception.InnerException == null)
                {
                    return string.Format(@"<form id='form1'>
                                <br>
                                <div class='container-fluid'>
                                    <label class='alert alert-danger col-sm-8'>
                                        <h4>Se ha detectado un error con la solicitud:</h4>
                                        <h4>{0}</h4>
                                    </label>
                                </div>
                            </form>", exception.Message);
                }
                MyTraceSource.myTraceSource.TraceEvent(TraceEventType.Error, 0, "Error al consultar en SCSP: '{0}'", exception.Message);
                throw new Exception("Error en la consulta SCSP.");
            }
        }
    }
}
```

Figura 10 - Código base de la integración con SCSP

Esta función es la que se encarga de paso a paso crear todo lo necesarios para poder devolver la información correcta. Lo primero que hace es crear la petición al servicio web de SCSP. Como se puede ver en el A.I, todas las peticiones tienen una parte genérica que es igual para todos los servicios. Esta parte genérica son los "Atributos" y las "Solicitudes", y se rellenan en otro fichero para mantener un orden y que el código sea más legible.

La primera función que se llama es “GetAtributos” y no tiene complicación alguna. Lo único que tenemos que rellenar en esta parte de la petición es el código de certificado. Este código es el que le indicará a SCSP que información queremos obtener.

```
/// <summary>
/// Rellena los atributos de la peticion según el codigo de servicio que se vaya a realizar
/// </summary>
/// <param name="codigoCertificado">Codigo de servicio especifico de SCSP</param>
/// <returns>Atributos de la transmision</returns>
public static Atributos GetAtributos(string codigoCertificado)
{
    Atributos atributos = new Atributos()
    {
        CodigoCertificado = codigoCertificado
    };

    return atributos;
}
```

Figura 11 - Función “GetAtributos”

La segunda función, “GetSolicitud”, es la que realmente tiene trabajo. Dentro de SCSP hay dos formas de pedir información, de modo síncrono o de modo asíncrono. Para este proyecto se decidió usar el modo síncrono porque no se necesitaba consultar mucha información a la vez.

```
public static SolicitudTransmision[] GetSolicitud(Expediente expediente, Contratacion.Tramitacion.Solicitudes.Solicitante solicitante,
    string codigoSCSP, string tipoDocumento, string documento)
{
    SolicitudTransmision[] solicitudes = new SolicitudTransmision[1];

    // Solo realizamos una peticion al ser sincrona
    solicitudes[0] = new SolicitudTransmision() {
        DatosGenericos = new SolicitudTransmisionDatosGenericos()
        {
            Solicitante = new Solicitante()
            {
                IdentificadorSolicitante = Properties.Settings.Default.IdentificadorSolicitante,
                NombreSolicitante = Properties.Settings.Default.NombreSolicitante,
                UnidadTramitadora = Properties.Settings.Default.UnidadTramitadora,
                Procedimiento = new Procedimiento()
                {
                    CodProcedimiento = Properties.Settings.Default.CodProcedimiento,
                    NombreProcedimiento = Properties.Settings.Default.NombreProcedimiento
                },
                Finalidad = Properties.Settings.Default.Finalidad,
                Consentimiento = Consentimiento.Si,
                Funcionario = new Funcionario()
                {
                    NombreCompletoFuncionario = SessionData.UserFileSystem.User.Name,
                    NifFuncionario = UserField.LoadUserField(SessionData.FileSystem, "NIF", "USUARIO").GetValue(SessionData.UserFileSystem.User)
                },
                IdExpediente = expediente.Id
            }
        },
        Item = null
    };

    if (codigoSCSP.Equals("SVDTUWS03")) // Si la consulta es de datos universitarios usamos los datos obtenidos en el formularo de InformesLicitadores.aspx
    {
        solicitudes[0].DatosGenericos.Titular = new ServiceSCSP.Titular
        {
            // Este servicio solo acepta CIF o NIF (NIF por defecto)
            TipoDocumentacion = (tipoDocumento == "NIE") ? TipoDocumentacion.NIE : TipoDocumentacion.NIF,
            TipoDocumentacionSpecified = true,
            Documentacion = documento,
        };
    }
    else // Si no, lo rellenamos con los datos del licitador directamente
    {
        // Parseamos el nombre completo
        AuxiliarSCSP.ParseNombreCompleto(solicitante.Nombre, out string nombreAux, out string apellido1Aux, out string apellido2Aux);

        solicitudes[0].DatosGenericos.Titular = new ServiceSCSP.Titular
        {
            TipoDocumentacion = TipoDocumentacion.NIF,
            TipoDocumentacionSpecified = true,
            Documentacion = solicitante.Documento,
            NombreCompleto = solicitante.Nombre,
            Nombre = nombreAux,
            Apellido1 = apellido1Aux,
            Apellido2 = apellido2Aux
        };
    }

    return solicitudes;
}
```

Figura 12 - Función "GetSolicitud"

La principal diferencia es que el modo síncrono solo crea una solicitud mientras que el asíncrono crea las que quieras, pero como en el código no tenemos una forma de diferenciar si es una petición síncrona o asíncrona, simplemente limitamos el tamaño del array de solicitudes a uno.

El tipo de variable "SolicitudTransmision" contiene dos atributos, "SolicitudTransmisionDatosGenericos" e "Item". El primero hace referencia a los datos genéricos de la solicitud, y el segundo indica la parte específica, que lo dejamos a "null" porque ninguno de los servicios que hemos desarrollado tiene datos específicos. Esto se puede ver en la figura 13 del A.I.

Dentro de los datos genéricos debemos rellenar dos campos, el solicitante y el titular. El solicitante hace referencia a la entidad que está pidiendo información, es decir, a la empresa que ha publicado la oferta y está mirando los licitadores que se han apuntado. La información de esta empresa está almacenada en un archivo de configuración, y esto es así porque esta aplicación la usarán muchos cliente de Pixelware, y cada uno tendrá sus parámetros definidos de una forma. Además de toda la información de la empresa también hay que indicar el usuario concreto que está llevando este proceso, "Funcionario" y el identificador del expediente en el que esté trabajando dicho "Funcionario". Todo esto es información relacionada con la empresa.

Ahora pasamos a la información relacionada con el licitador, que es el campo titular de los datos genéricos. En este momento tenemos dos casos separados:

1. Están pidiendo información acerca de los títulos universitarios de alguien.
2. Cualquier otro servicio.

¿Por qué diferenciarlos? Como ya he comentado antes, no tendría sentido pedir títulos universitarios de un licitador si este es una empresa. Entonces, si recibimos el código de SCSP correspondiente a la solicitud de títulos universitarios por documentación, rellenamos los campos del titular con el documento que se especificó en el modal.

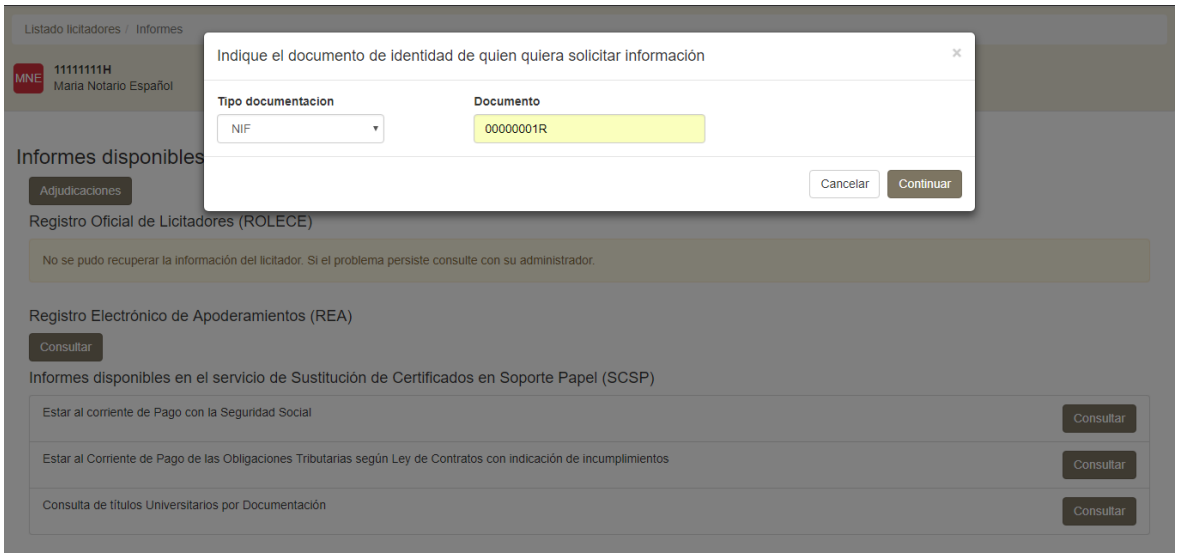

Figura 13 - Documento especificado en el modal

Si nos encontramos en el punto 2, es decir, que hemos recibido la solicitud de información de cualquier otro servicio, entonces rellenamos la información con el solicitante que habíamos obtenido en el DataService.svc.

En este momento nos encontramos con una necesidad de crear una función capaz de devolver el nombre de una persona dividida en nombre, apellido1, apellido2. Esta función era necesaria porque cada licitador guarda su nombre de una forma distinta, y no hay un estándar que podamos seguir.

```
public static void ParseNombreCompleto(string nombreCompleto, out string nombre, out string apellido1, out string apellido2)
{
    // Si contiene ',' probablemente esté escrito Apellido1 Apellido 2, Nombre
    if (nombreCompleto.Contains(','))
    {
        // Quitamos la coma para facilitar el split
        nombreCompleto.Remove(nombreCompleto.IndexOf(','), 1);

        string[] aux = nombreCompleto.Split(' ');
        // Si tiene más de 3 probablemente sea un nombre compuesto
        if (aux.Count() > 3)
        {
            nombre = String.Format("{0} {1}", aux[2], aux[3]);
            apellido1 = aux[0];
            apellido2 = aux[1];
        }
        // Si tiene tamaño 3 entonces hay nombre y dos apellidos
        else if (aux.Count() == 3)
        {
            nombre = aux[2];
            apellido1 = aux[0];
            apellido2 = aux[1];
        }
        // Si tiene tamaño 2, probablemente tenga nombre y apellido1
        else
        {
            nombre = aux[1];
            apellido1 = aux[0];
            apellido2 = "";
        }

    }
    // Nombre Apellido1 Apellido2
    else
    {
        string[] aux = nombreCompleto.Split(' ');
        // Si tiene más de 3 probablemente sea un nombre compuesto
        if (aux.Count() > 3)
        {
            // Deolvemos el nombre
            nombre = String.Format("{0} {1}", aux[0], aux[1]);
            apellido1 = aux[2];
            apellido2 = aux[3];
        }
        // Si tiene tamaño 2 significa que tiene ambos apellidos
        else if (aux.Count() == 3)
        {
            nombre = aux[0];
            apellido1 = aux[1];
            apellido2 = aux[2];
        }
        // Si tiene tamaño dos entonces solo tiene el primer apellido
        else if (aux.Count() == 2)
        {
            nombre = aux[0];
            apellido1 = aux[1];
            apellido2 = "";
        }
        // Si no, tiene solo nombre
        else
        {
            nombre = aux[0];
            apellido1 = "";
            apellido2 = "";
        }
    }
}
```

Figura 14 - Función que obtiene el nombre, apellido1 y apellido2 dependiendo del formato

En esta función se recogen los casos más comunes con los que los usuarios guardan sus nombres.

Una vez terminada la solicitud, la función “GetSolicitud” la devuelve y “ObtenerInformeConsultaSCSP” la guarda dentro de la petición que se mandará a SCSP.

Una vez creada la petición, hay que mandársela a SCSP a través de su cliente de servicio web. Para configurar este servicio web hace falta crear una conexión al endpoint de SCSP (esto nos produjo ciertos problemas que comentaré en la sección de pruebas), después hay que abrir la conexión y mandar la petición. En este momento es en el único en el que diferenciamos entre síncrona y asíncrona. Una vez recibida la información cerramos la conexión y pasamos la función de parseo de la parte específica de la respuesta.

La captura de pantalla de esta función está en el A.III ya que es demasiado grande para ponerla aquí.

En esta función lo que hacemos es comprobar los códigos de retorno de SCSP que dependen de cada servicio (se pueden ver en [5][6][7]). Si el código recibido es de error, le mostramos al cliente que parámetro ha introducido mal para que vuelva a intentarlo, si no, formamos un documento HTML con todos los datos del licitador del que ha pedido información y le añadimos el resultado SCSP. Este HTML es el que más tarde, mediante Jquery se introducirá en la pestaña nueva del navegador.

Para poder realizar el parseo, tuvimos que pasar un fichero XML a clases de C#, estas clases están dentro de la carpeta “Respuestas” de la estructura mostrada en la figura 4. Tenemos tres clases, una por servicio:

- RespuestaTGSS.
- RespuestaAEAT.
- RespuestaEducacionDocumento.

Cada una de ellas contiene la estructura exacta de los datos específicos que se ven en las figuras del A.I (A.I.1A.I.2A.I.3). Esta estructura tiene que ser idéntica al mensaje, porque si no a la hora de pasar el XML no detectará ciertas etiquetas y dará errores. Voy a añadir una captura de RespuestaTGSS para que se vea la estructura, pero las otras dos clases son iguales solo que con los datos que ese servicio devuelve.

Los elementos claves para que la deserialización de un XML funcione son las etiquetas que se le añaden a las distintas clases justo antes de su declaración. Tenemos la etiqueta “[Serializable ()]” que indica que esa clase puede ser usada por una función que pase un XML a C#. Además, como ya comentaré en el apartado de pruebas, tuvimos que añadir el elemento “[XmlRoot ("Retorno…")]” el cual indica que el fichero XML empieza por la etiqueta llamada “Retorno”. Sin esta etiqueta el deserializador no sabe por dónde empezar y no hace su trabajo. Luego observamos las distintas jerarquías de etiquetas, donde cada clase que se declara significa una nueva rama. Solo aquellos atributos que son de tipo elemental (String, decimal, etc.) indican una hoja, y entonces se añade el valor que se encuentre en el XML. Cada clase debe declararse con la etiqueta [XmlElement ()], que simplemente indica que es un elemento a rellena al leer el XML. Se puede añadir un nombre para asegurarnos de que escogemos bien el nodo XML y además un “Order” para tener la certeza de que se rellenan todos los campos en el orden en el que vienen en el XML.

```
namespace ContratacionApi.Integraciones.SCSP.Utils
{
    [Serializable()]
    [XmlRoot("Retorno", Namespace = "http://intermediacion.redsara.es/scsp/esquemas/datosespecificos")]
    public class RespuestaTGSS
    {
        private Retorno _retorno;

        [XmlElement("Retorno", Order = 0)]
        public Retorno Retorno
        {
            get
            {
                return this._retorno;
            }
            set
            {
                this._retorno = value;
            }
        }
    }

    public class Retorno
    {
        private string _codigo;

        private string _resultado;

        private string _descripcion;

        [XmlElement("Codigo", Order = 0)]
        public string Codigo
        {
            get
            {
                return this._codigo;
            }
            set
            {
                this._codigo = value;
            }
        }

        [XmlElement("Resultado", Order = 1)]
        public string Resultado
        {
            get
            {
                return this._resultado;
            }
            set
            {
                this._resultado = value;
            }
        }

        [XmlElement("Descripcion", Order = 2)]
        public string Descripcion
        {
            get
            {
                return this._descripcion;
            }
            set
            {
                this._descripcion = value;
            }
        }
    }
}
```

Figura 15 - Datos específicos guardados en RespuestaTGSS

Poco más hay que comentar de estas clases. De por sí no hacen nada en especial solo son clases auxiliares para usarlas en la creación del resultado.

Una vez vistas las clases, pasemos al parseo del XML, que se lleva a cabo en la función del A.III.

```
serializer = new XmlSerializer(typeof(RespuestaEducacionDocumento));

RespuestaEducacionDocumento especificosEducacionDocumentos = (RespuestaEducacionDocumento)serializer.Deserialize(
    new StringReader(((XmlNode[])respuesta.Transmisiones[0].Item)[1].OuterXml));
```

Figura 16 - Deserialize de la respuesta de títulos universitarios por documentación

Este es un ejemplo de cómo se pasa del XML de la respuesta de SCSP a la clase "RespuestaEducacionDocumento". Aquí tuvimos otro problema, que también comentaré en el apartado de pruebas, porque usábamos el "InnerXml" de la respuesta en vez del "OuterXml", que es el que contiene todas las etiquetas sin modificar.

La función de "Deserialize ()[12]" es propia de .NET y recibe como parámetro un "Stream[13]" que contiene una cadena de caracteres que con todo el XML. Este XML viene en la respuesta de SCSP, pero viene como una clase "Object", es decir, no es un string por lo que no podemos usarlo directamente. Para ello le hacemos un "casting" al tipo de .NET "XmlNode [][14]" el cual nos permite acceder al código XML de un "Object".

Tras esto tenemos en la variable "especificosEducacionDocumentos" todos los valores del XML y podemos acceder a ellos como cualquier otra clase en la programación orientada a objetos.

En resumen, la función de parseo lee el XML, lo pasa a la clase correspondiente según el servicio que haya elegido el usuario, y a partir de ahí forma el mensaje HTML que devolverá. Se puede ver en el A.III el HTML que se va formando.

[12] https://msdn.microsoft.com/es-es/library/dsh84875(v=vs.110).aspx

[13] https://msdn.microsoft.com/es-es/library/system.io.stream(v=vs.110).aspx

[14] https://msdn.microsoft.com/es-es/library/system.xml.xmlnode(v=vs.110).aspx

5 Integración, pruebas y resultados

En este apartado voy a comentar las diferentes pruebas que se realizaron para asegurar que la aplicación funcionaba correctamente y como el cliente quería. Dentro de cada tipo de prueba añadiré los resultados de las pruebas.

Las capturas de pantalla de los resultados de las pruebas se pueden ver en el A.II.

5.1 Pruebas de integración

En este apartado de pruebas no trabajé demasiado dado que yo estaba desarrollando un evolutivo sobre una aplicación de la empresa Pixelware. Durante todo el desarrollo yo tenía montada la aplicación en un entorno local, donde yo podía probar cualquier cosa, así que si llegué a hacer pruebas de integración, pero dada la estructura de la empresa ese no era mi trabajo.

Yo desarrollé la aplicación y le realice una serie de pruebas que comentaré ahora después, después de eso, el equipo de calidad se encargó de montar la aplicación en un entorno de desarrollo real y realizar más pruebas de integración.

Todas esas pruebas pasaban por mí una vez realizadas, ya que era el encargado de corregirlas. Una de las pruebas fallaba porque el conector que teníamos entre nuestra aplicación y el servicio web de SCSP que usábamos, fallaba. Realmente no era un error nuestro, ya que era problema de la red, aunque sí que es verdad que pudimos mejorar la forma de abrir la conexión con el servicio web añadiendo un "endpoint[15]" a la configuración de nuestra aplicación.

Otro error que detectamos, fue a la hora de juntar el Frontend y el Backend, ya que desde el servidor devolvíamos un HTML puro que se tenía que plasmar en una página de resultados. Este HTML lo añadimos con Jquery a una pestaña nueva en el navegador del cliente. Pues el error venía de que no recibía el HTML bien, y eso producía que el resultado apareciese en blanco. Esto lo arreglamos cambiando la forma en la que cargábamos ese HTML en la nueva página además de añadiendo al HTML que se generaba en el servidor algunas etiquetas que faltaban.

Quitando esos problemas, las demás pruebas de integración fueron exitosas, dejando este apartado sin muchos más problemas.

5.2 Pruebas unitarias

Dentro de la aplicación podemos distinguir entre el Frontend y el Backend, pero a su vez el Backend se divide en distintos módulos:

- Servicio web implementado, el cual comunica el cliente con el servidor, y el servidor con el módulo que implementa SCSP.
- Módulo encargado del parseo de la respuesta de SCSP, pasando del mensaje XML recibido al HTML que se le pasa al servicio web integrado.

[15] Dirección final de una aplicación a nivel de transporte.

- Integración con SCSP. Este módulo es que crea las peticiones con los datos recibidos del servicio web, abre la conexión con el cliente de SCSP y manda/recibe los datos.

Estos tres módulos se probaron por separado, teniendo especial cuidado en el paso de argumento entre módulos, ya que ningún argumento podía ser modificado, ya que esto llevaría a recibir una información errónea a la que el cliente había solicitado.

El primer módulo no tiene una lógica muy complicada, simplemente recibe unos parámetros, saca unos valores de la base de datos y llama al módulo de SCSP. Dado que no tiene una carga grande de código, realizamos pruebas de caja negra para asegurarnos que recibimos la información que habíamos pedido y poco más. Por esto, los problemas encontrados en este módulo no eran propios, es decir, eran errores que venían del resto de módulos. Aunque eso no significa que no se integrasen bien, simplemente que los demás módulos conectados no realizaban su función con exactitud. En resumen, el módulo del servicio web no recibió mucha intención al ser muy básico, aunque importante.

El siguiente módulo es el más importante. La integración de SCSP tiene mucha lógica por debajo y muy importante. Es el módulo en el que según que opción haya elegido el cliente, se ejecutará de una forma u otra el código para recibir la distinta información de SCSP.

Como en el otro módulo, se realizaron pruebas de caja negra. Estas pruebas salieron mal al principio ya que, como he comentado en las pruebas de integración, la conexión con SCSP se hacía mal y fallaba, entonces recibíamos constantemente un error que decía que la conexión no se había realizado con éxito. Este error no era detectable a simple vista, ya que el código está programado con excepciones, así que se devolvía un error genérico que no daba mucha información. Al encontrarnos con esto, tuvimos que pasar a las pruebas de caja blanca.

En estas pruebas, depuramos todo el código probando todas las opciones posibles que permitíamos desde el lado del cliente, y ahí fue cuando descubrimos que la conexión estaba fallando. Añadimos al fichero de configuración de la aplicación el endpoint del servicio de SCSP. A parte del error de conexión, salieron varios fallos a la hora de rellenar la petición a SCSP, porque algunos campos que se sacaban de la BD no tenían el formato correcto. También comentar que los distintos caminos que se podían tomar en el código (en base a los “if” que había de filtrado) estaban bien especificados y no quedaba nunca un camino suelto sin hacer nada.

Por último, hablar del módulo encargado de pasar la respuesta del servidor de SCSP a HTML. Las pruebas de caja negra fueron rápidas, ya que si no recibíamos los datos exactos que se nos especificaban en los casos de prueba, el módulo funcionaba mal. El primer error fue que al abrirse la pantalla de los resultados, siempre estaba en blanco. Esto ocurrió a unos fallos en el paso de XML a HTML, pues faltaban etiquetas HTML por poner, y además el método de Jquery para cambiar el contenido HTML de una ventana no funcionaba como esperábamos.

Otro error, que nos dio bastantes problemas fue la función de parseo. El leer un XML y obtener todos los atributos dentro de él y guardarlos en una clase de C#. Aquí detectamos varios errores:

- Las clases que iban a contener los datos del XML de respuesta no se declararon de la manera correcta para que pudieran albergar datos XML. El problema estaba en que no definíamos las clases como XMLRoot[16], por lo que el parser no sabía dónde empezar a añadir atributos.
- El otro error que detectamos era que no se cogía bien el XML. La respuesta de SCSP tiene un campo de DatosEspecificos, que es un XML guardado como un String. Pero a la hora de recibir la respuesta, ese XML no lo sacábamos del lado correcto, y faltaban etiquetas XML para que el parseador lo detectase. Este error no tenía mucho misterio, pero influía mucho en el funcionamiento de la aplicación en general porque sin eso, el resultado que se mostraba siempre salía en blanco.

Estas fueron las últimas pruebas que realizamos, porque a partir de ahí funcionaba correctamente a nivel de código.

5.3 Pruebas de diseño gráfico

He decidido añadir estas pruebas, porque aunque no las hice yo, sí que las corregí y me parecen unas pruebas importantes en este proyecto ya que en todo momento lo que hacía tenía unos requisitos del cliente detrás.

Básicamente eran pruebas en las páginas web de la aplicación, que lo que buscaban era dejar el producto lo más parecido posible a lo que el cliente había pedido. Había que mover botones de sitio, cambiar colores y tamaños de letra etc. Creo que son cambios no importantes a nivel de esfuerzo, pero sí a nivel de empresa, ya que hay que darle el producto al cliente lo mejor posible y que se acerque más a lo que él pidió.

[16] https://stackoverflow.com/questions/364253/how-to-deserialize-xml-document/387566#387566

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Durante todo el desarrollo he tenido que estar trabajando en un ambiente de empresa, en el que tenía a compañeros que me ayudaban y a los que tenía que ayudar, muchas veces tenía que esperar información de otras personas para poder seguir y mantener el proyecto por la línea que se esperaba y sobre todo mantener un contacto con el cliente para asegurarme de que todo salía bien.

Creo que el haber desarrollado mi TFG en ese entorno ha sido una diferencia notoria respecto al haberlo hecho en la universidad, porque aquí tenía la presión de que era mi TFG pero a la vez era un desarrollo para un cliente.

Sin duda una experiencia satisfactoria que me ha dejado con buen sabor de boca, estoy muy a gusto con el trabajo que he realizado y cómo ha quedado el producto final que hemos presentado.

6.2 Trabajo futuro

La ley española sigue cambiando, y las empresas que trabajan con ella no pueden permitirse el lujo de parar. La aplicación que hemos desarrollado satisfacía las necesidades de un cliente, pero cada cliente tiene unas necesidades distintas.

A corto plazo, tenemos en mente aumentar los servicios ofrecidos en esta aplicación, porque solo hemos desarrollado tres, pero SCSP tiene muchos más y que son muy útiles en la administración pública.

También tenemos en mente añadir más servicios similares a SCSP, como por ejemplo:

- GoVa, un servicio para crear números de expedientes únicos en el gobierno vasco.

- Rolece, registro oficial de licitador y empresas clasificadas por el estado, contiene los datos de personalidad y capacidad de obrar, autorizaciones y habilitaciones, solvencia y clasificación empresarial.

Por último añadir una funcionalidad propia de Pixelware, que no requiere ningún servicio externo. Esta funcionalidad permite a un cliente conocer todas las adjudicaciones de una empresa que haya presentado su oferta, pudiendo filtrar por tipo de expediente, fecha de adquisición etc. Además de mostrar una tabla con todas las adjudicaciones y una gráfica dinámica que muestra adjudicaciones en €/mes.

Referencias

[1] Jorge García Barderas, “CA_P_000509 3.1.3 - Sustitución de Certificados en Soporte Papel SCSP”, Noviembre 2017.

[2] Oficina Técnica SCSP, “Catalogo Servicios de Verificación y consulta de Datos SCSP”, Marzo 2017.

[3] Oficina Técnica SCSP, “Manual de configuración del Recubrimiento WS SCSP J2EE V3.4.3”, Junio 2015.

[4] Oficina Técnica SCSP, “Manual de Usuario del Recubrimiento SCSP J2EE 3.4.3”, Junio 2015.

[5] Oficina Técina SCSP, “Servicios de Consulta Estar al Corriente de Obligaciones Tributarias. Agencia Tributaria”, Agosto 2017.

[6] Oficina Técina SCSP, “Servicio de Consulta de Títulos Universitarios (Consulta por documentación). Ministerio de Educación”, Abril “017.

[7] Oficina Técina SCSP, “Servicio de Consulta de Estar al Corriente de Pago de Obligaciones con TGSS. Tesorería General de la Seguridad Social”, Febrero 2017.

[8] DGIAE, “Protocolo de Sustitucion de Certificados en Soporte Papel”, Noviembre 2011.

Glosario

API	Application Programming Interface
SCSP	Sustitución de Certificados Sobre Papel
TGSS	Tesorería General de la Seguridad Social
AEAT	Agencia Estatal de Administración Tributaria
Frontend	Parte de la aplicación que se ejecuta en el lado del cliente.
Backend	Parte de la aplicación que se ejecuta en el lado del servidor.
JA	Javascript.
SO	Sistema operative.
BD	Base de datos.
RF	Requisito functional.
RNF	Requisito no functional.

Anexos

I. *Diagrama de mensajes de Servicio*

A continuación se muestran los diagramas de atributos que contendrán los mensajes tanto de petición como de respuesta de los distintos servicios implementados [2].

Los “DatosGenericos” de la petición y la respuesta son iguales en todos los servicios, y por eso no incluyo su diagrama en ningún servicio.

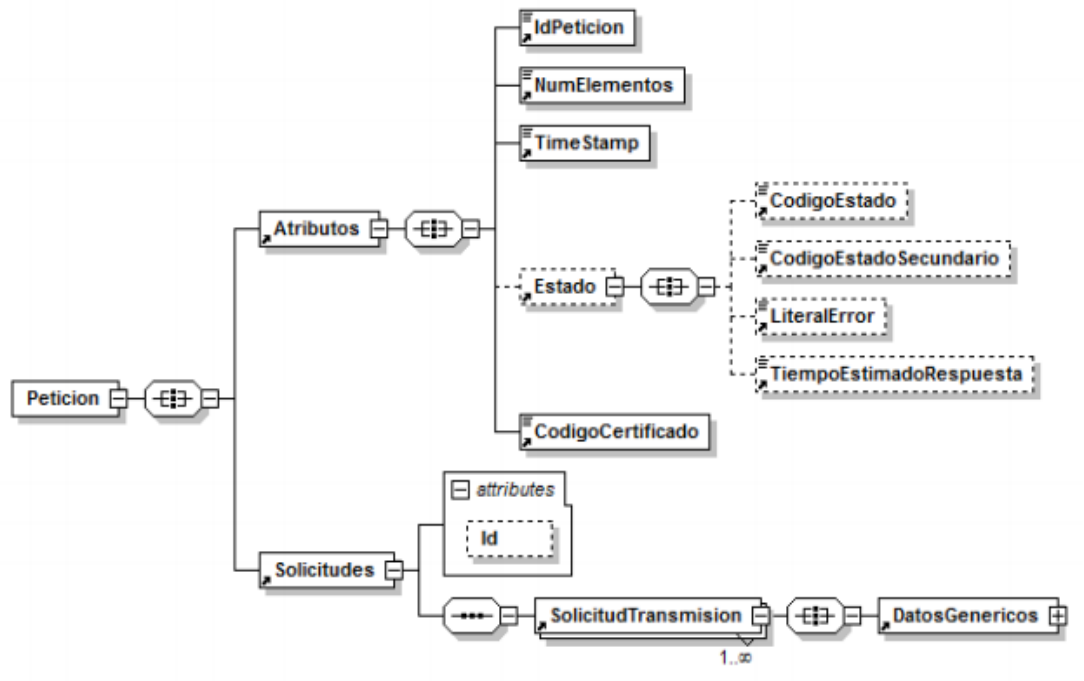

Figura 17 - Diagrama del menaje de petición a cualquier servicio

Figura 18 - Diagrama de la rama de DatosGenericos del mensaje de petición del servicio

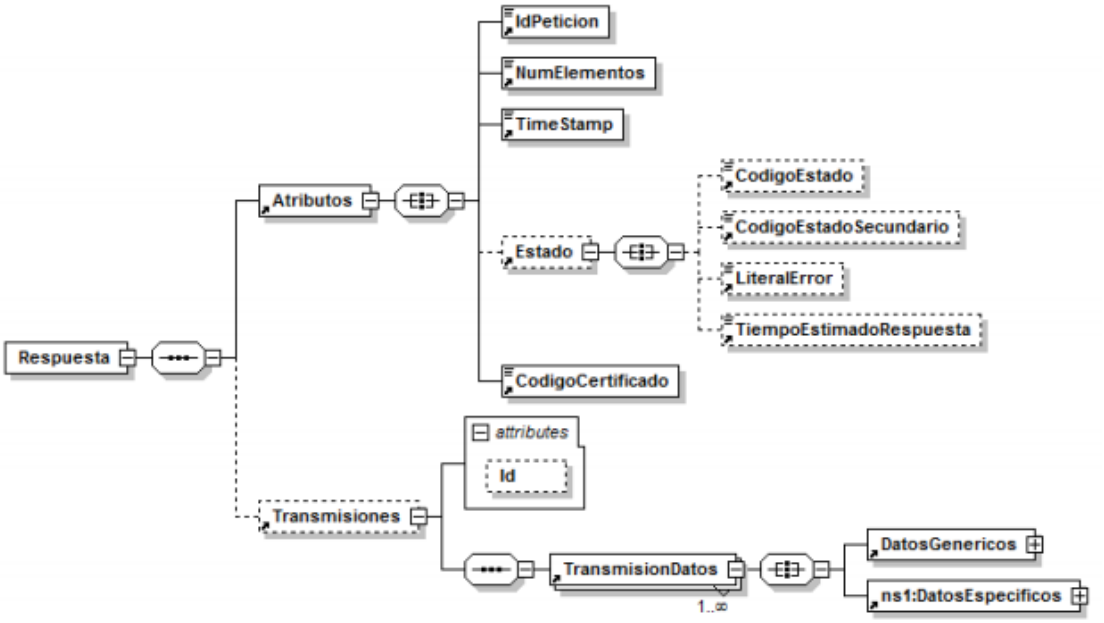

Figura 19 - Diagrama del mensaje de respuesta de cualquier servicio

Figura 20 - Diagrama de la rama de DatosGenericos del mensaje de respuesta del servicio

Sin embargo, los “DatosEspecificos” de la petición y la respuesta, dependen de cada servicio, pero en nuestro caso, ninguno de los tres servicios integraciones tiene “DatosEspecificos” en la petición, solo en la respuesta del servicio.

I.1 TGSS

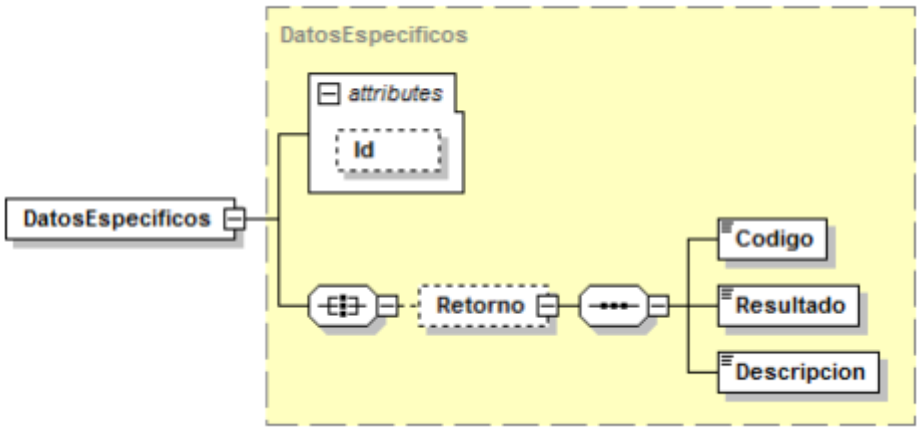

Figura 21 - Diagrama de DatosEspecíficos de Respuesta del Servicio de TGSS

I.2 AEAT

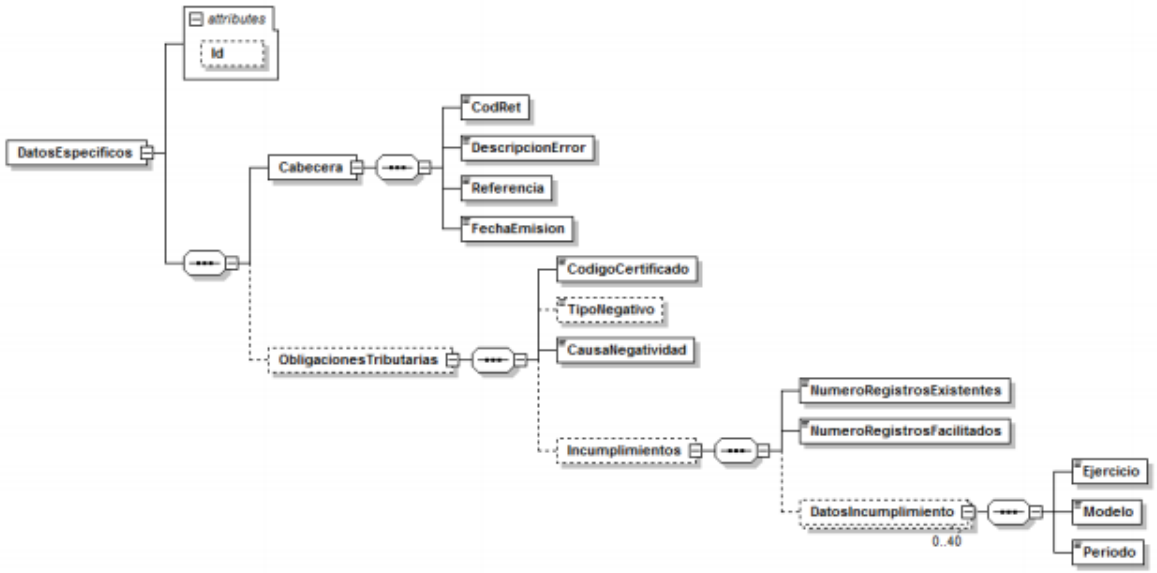

Figura 22 - Diagrama de DatosEspecíficos de Respuesta del Servicio de AEAT

I.3 Títulos universitarios

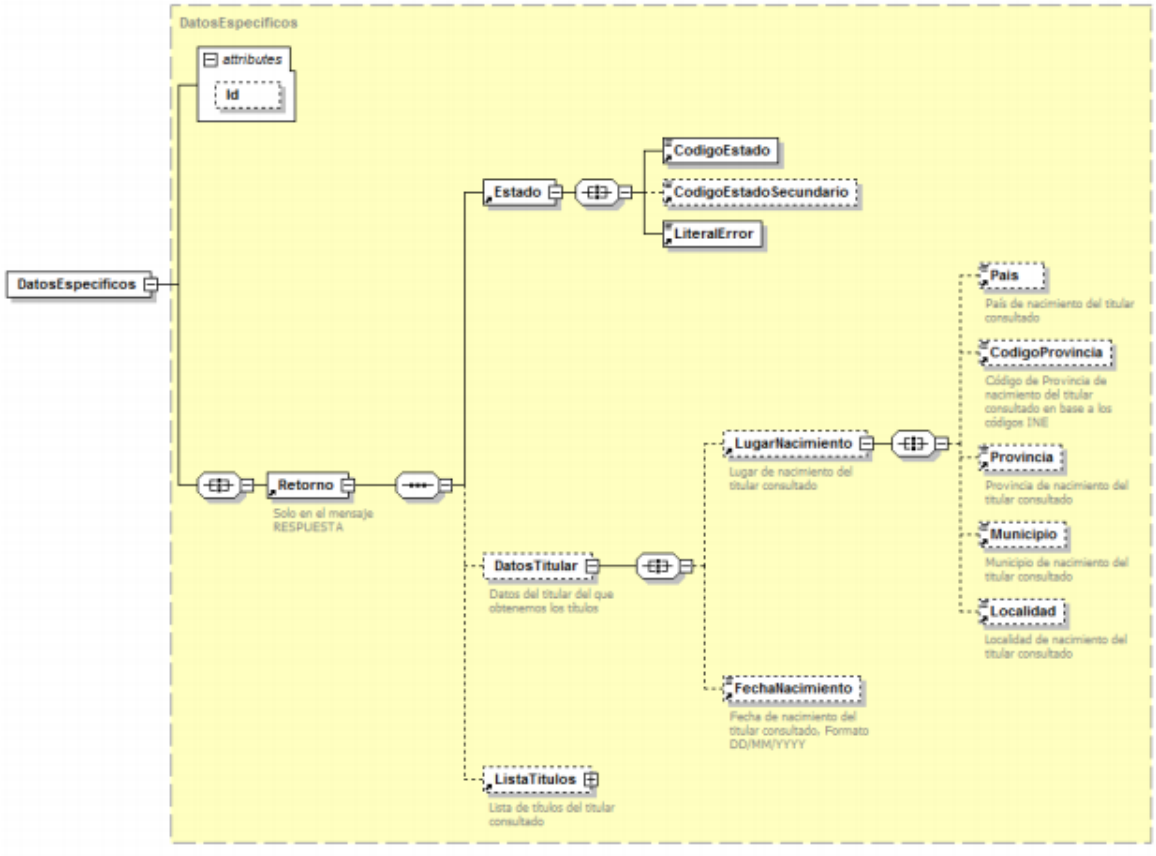

Figura 23 - Diagrama de DatosEspecíficos de Respuesta del Servicio de Títulos Universitarios

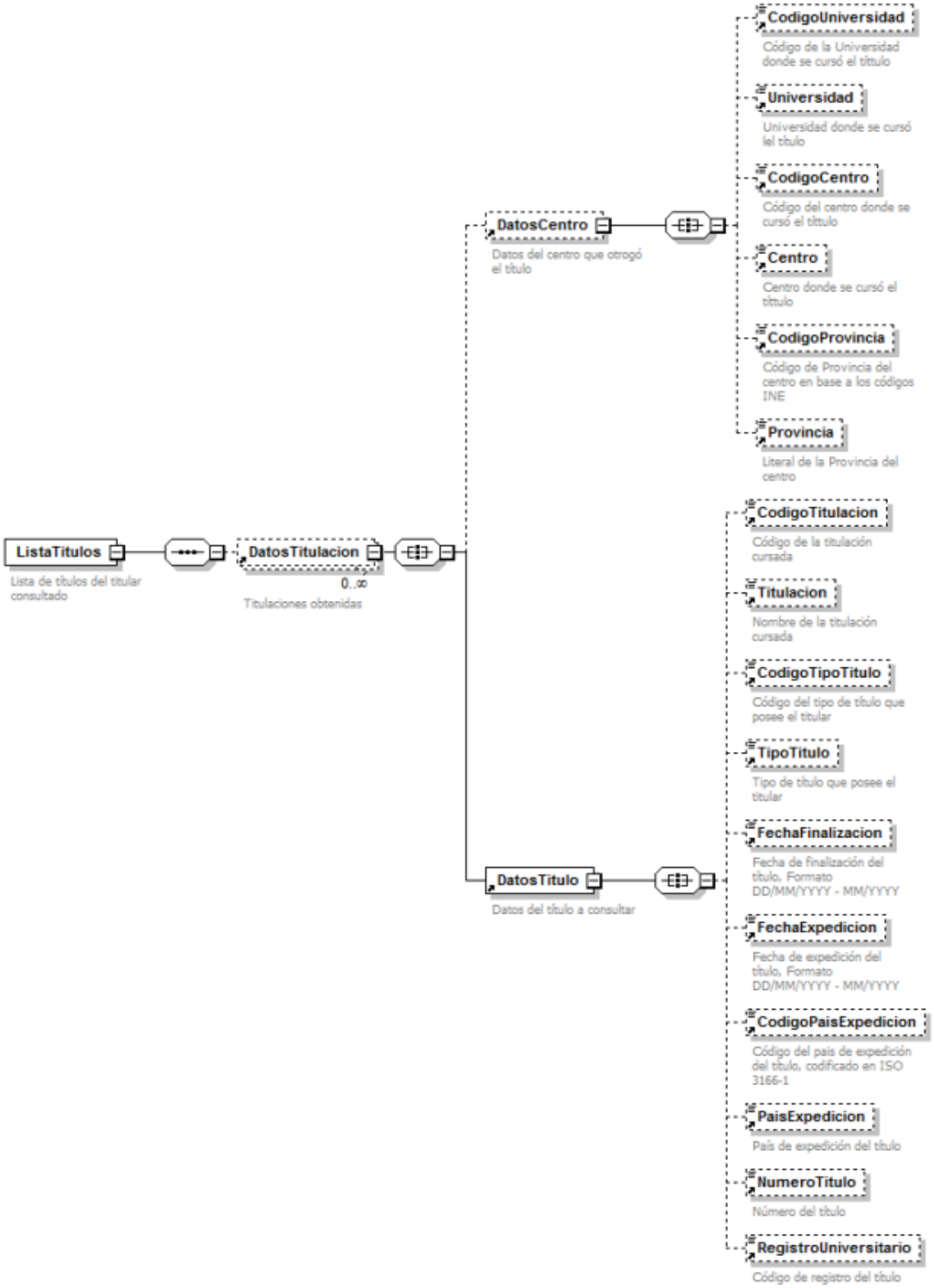

Figura 24 - Detalle de la rama ListaTitulos de los Datos Específicos de la respuesta

II. Resultados

En este anexo voy a incluir todas las capturas de pantalla de los resultados obtenidos.

La primera captura es del mensaje que se muestra cuando hemos solicitado información sobre algún licitador, pero este no consta en la base de datos de SCSP.

Figura 25 - Titular no identificado

El siguiente mensaje muestra un error a la hora de realizar alguna solicitud. Suele ser causado por un problema del servicio de SCSP o de la conexión entre el servidor de nuestra aplicación y el de SCSP.

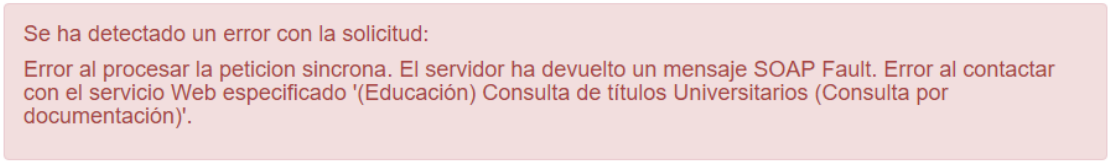

Figura 26 - Error en la solicitud a SCSP

La última captura es el resultado a la petición de títulos universitarios por documento de identidad. Para este caso de prueba usamos un documento que nos proporcionó SCSP para que supiesemos que el resultado era el correcto.

MARGARITA MONTSERRAT GARRIDO ESPINOSA (00000001R)

Nº título	Titulación	Tipo titulación	Centro	Universidad	Fecha finalización	Fecha expedición	País expedición	Registro universitario
2005113280	Licenciado en Bellas Artes (3021)	Licenciado (04)	Facultad de Bellas Artes (28027126)	Universidad Complutense de Madrid (10)	01/06/2004	10/09/2004	ESPAÑA (3021)	724
1999029433	Ingeniero de Telecomunicación (1008)	Ingeniero (06)	Escuela Técnica Superior de Ingenieros de Telecomunicación (28026951)	Universidad Politécnica de Madrid (25)	01/04/1997	09/06/1997	ESPAÑA (1008)	724

* Los números entre paréntesis corresponden a los códigos asociados a cada valor.

Figura 27 - Resultado de títulos universitarios por documento de identidad

III. Función “ParseRespuestaSCSPToHtml”

```
public static string ParseRespuestaSCSPToHtml(Respuesta respuesta, string codigoSCSP)
{
    XmlSerializer serializer;
    string header = "";
    string body = "";
    string html = "";

    MyTraceSource.myTraceSource.TraceEvent(TraceEventType.Verbose, 0, "Empieza el parseo de los datos especificos de la respuesta con codigoSCSP {0}", codigoSCSP);
    if (respuesta.Atributos.Estado.CodigoEstado == "0003")
    {
        switch (codigoSCSP)
        {
            // Titulos universitarios por documentacion
            case "SVDTUWS03":
                serializer = new XmlSerializer(typeof(RespuestaEducacionDocumento));

                RespuestaEducacionDocumento especificosEducacionDocumentos = (RespuestaEducacionDocumento)serializer.Deserialize(
                    new StringReader(((XmlNode[])respuesta.Transmisiones[0].Item)[1].OuterXml));

                // Si el resultado devuelto, contiene al menos 1 titulo, mostramos la tabla
                if (especificosEducacionDocumentos.ListaTitulos != null)
                {
                    // Cabecera de la tabla de resultados
                    header = string.Format(@"
                        <form id='form1'>
                        <br>
                        <div class='container-fluid'>
                        <h4>{0} ({1})</h4>
                        <table class='table table-sm table-striped'>
                          <thead>
                            <tr>
                              <th scope='col'>Nº titulo</th>
                              <th scope='col'>Titulación</th>
                              <th scope='col'>Tipo titulación</th>
                              <th scope='col'>Centro</th>
                              <th scope='col'>Universidad</th>
                              <th scope='col'>Fecha finalización</th>
                              <th scope='col'>Fecha expedición</th>
                              <th scope='col'>País expedición</th>
                              <th scope='col'>Registro universitario</th>
                            </tr>
                          </thead>
                          <tbody>", respuesta.Transmisiones[0].DatosGenericos.Titular.NombreCompleto,
                                    respuesta.Transmisiones[0].DatosGenericos.Titular.Documentacion);

                    foreach (DatosTitulacion titulo in especificosEducacionDocumentos.ListaTitulos.DatosTitulacion)
                    {
                        // Cuerpo de la tabla
                        body = string.Format(@"
                            <tr>
                              <th scope='row'>{0}</th>
                              <td>{1} ({2})</td>
                              <td>{3} ({4})</td>
                              <td>{5} ({6})</td>
                              <td>{7} ({8})</td>
                              <td>{9}</td>
                              <td>{10}</td>
                              <td>{11} ({2})</td>
                              <td>{12}</td>
                            </tr>", titulo.DatosTitulo.NumeroTitulo, titulo.DatosTitulo.Titulacion, titulo.DatosTitulo.CodigoTitulacion, titulo.DatosTitulo.TipoTitulo, titulo.DatosTitulo.CodigoTipoTitulo,
                                    titulo.DatosCentro.Centro, titulo.DatosCentro.CodigoCentro, titulo.DatosCentro.Universidad, titulo.DatosCentro.CodigoUniversidad, titulo.DatosTitulo.FechaFinalizacion,
                                        titulo.DatosTitulo.FechaExpedicion, titulo.DatosTitulo.PaisExpedicion, titulo.DatosTitulo.CodigoPaisExpedicion, titulo.DatosTitulo.RegistroUniversitario);

                        // Añadimos la fila del titulo al html final
                        header = string.Concat(header, body);
                    }

                    header = string.Concat(header, "</tbody></table><div><h6>* Los números entre paréntesis corresponden a los códigos asociados a cada valor.</h6></div></div></form>");
                    html = string.Concat(html, header);
                }
                // Si no contiene titulos, mostramos un mensaje informando que no se ha recibido nada.
                else
                {
                    // Generamos el mensaje de error
                    header = string.Format("<form id='form1'><br><div class='container-fluid'><label class='alert alert-info col-sm-6'>");
                    html = string.Format("<h4>No se han recibido titulos universitarios asignados al documento {0}</h4>", respuesta.Transmisiones[0].DatosGenericos.Titular.Documentacion);
                    html = string.Concat(header, string.Concat(html, "</label></div></form>"));
                }

                break;
            // Corriente pago via AEAT
            case "EC0T101I":
                serializer = new XmlSerializer(typeof(RespuestaAEAT));

                RespuestaAEAT especificosAEAT = (RespuestaAEAT)serializer.Deserialize(
                    new StringReader(((XmlNode[])respuesta.Transmisiones[0].Item)[1].OuterXml));

                // HTML que muestra el resultado de la peticion
                body = string.Format(
                    @"<form id='form1'>
                        <br>
                        <div class='container-fluid col-sm-7'>
                            <span class='clearfix' style='background-color: white;'>
                                <h4>{0} ({1})</h4>
                            </span>
                            <span class='clearfix' style='background-color: white;'>
                                <span class='media-left media-middle'>
                                    {2}
                                </span>
                                <span class='media-left media-middle'>
                                    Número de incumplimientos: {3}
                                </span>
                            </span>
                        </div>
                    </form>", respuesta.Transmisiones[0].DatosGenericos.Titular.NombreCompleto,
                                respuesta.Transmisiones[0].DatosGenericos.Titular.Documentacion,
                                    especificosAEAT.ObligacionesTributarias.CausaNegatividad,
                                        especificosAEAT.ObligacionesTributarias.Incumplimientos);

                // Concateno el body del html al html general
                html = string.Concat(html, body);
                break;
            // Corriente pago via TGSS
            case "Q2827003ATGSS001":
                serializer = new XmlSerializer(typeof(RespuestaTGSS));

                RespuestaTGSS especificosTGSS = (RespuestaTGSS)serializer.Deserialize(
                    new StringReader(((XmlNode[])respuesta.Transmisiones[0].Item)[1].OuterXml));

                // HTML que muestra el resultado de la peticion
                body = string.Format(
                    @"<form id='form1'>
                        <br>
                        <div class='container-fluid col-sm-7'>
                            <h4>{0} ({1})</h4>
                            <span class='clearfix' style='background-color: white;'>
                                <span class='media-left media-middle'>
                                    {2}
                                </span>
                            </span>
                        </div>
                    </form>", respuesta.Transmisiones[0].DatosGenericos.Titular.NombreCompleto,
                                respuesta.Transmisiones[0].DatosGenericos.Titular.Documentacion,
                                    especificosTGSS.Retorno.Descripcion);
```

```
                // Concateno el body del html al html general
                html = string.Concat(html, body);
                break;
            // Aquí no se debería llegar nunca ya que los códigos los damos nosotros mismos
            default:
                MyTraceSource.myTraceSource.TraceEvent(TraceEventType.Error, 0, "El codigo SCSP '{0}' es erróneo", codigoSCSP);
                return null;
        }
    }
    else
    {
        // Generamos un mensaje de error que muestre el error recibido de SCSP
        header = string.Format("<form id='form1'><br><div class='container-fluid'><label class='alert alert-warning col-sm-8'>");
        html = string.Format("<h4>Error {0}: {1}</h4>", respuesta.Atributos.Estado.CodigoEstado, respuesta.Atributos.Estado.LiteralError);
        html = string.Concat(header, string.Concat(html, "</label></div></form>"));
    }

    MyTraceSource.myTraceSource.TraceEvent(TraceEventType.Verbose, 0, "Termina el parseo de los datos especificos de la respuesta con codigoSCSP {0}", codigoSCSP);

    return html;
}
```

Figura 28 - Función auxiliar "ParseRespuestaSCSPToHtml[17]"

[17] Las tres capturas son de la misma función.