Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una solución de integración entre sistemas bancarios e intermediarios

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Carlos Negrillo Pastor

Tutor: Pedro José Valderas Aranda

2018-2019

Resumen

En la actualidad, la integración entre distintos sistemas está a la orden del día. Concretamente el trabajo realizado, se ha enfocado en la integración del sector bancario y de las finanzas. Los objetivos perseguidos son, por un lado, conseguir simplificar y mejorar las comunicaciones, y por otro lado, estandarizar y transformar toda la información, de manera que fuese fácilmente legible para el receptor.

Se plantea un escenario inicial formado por bancos e intermediarios, donde los bancos son quienes comparten la información y los intermediarios quienes desean recibirla. Partiendo de esta premisa, nuestro proyecto propone un sistema de "suscripciones", donde cada intermediario mediante una web Responsive, será capaz de seleccionar todos aquellos bancos de los que desea recibir información. Además, el intermediario tendrá, por un lado, la posibilidad de elegir el formato en que recibe la información (XML, JSON…), y, por otro lado, podrá obtener una conversión customizada de la sintaxis y la lógica, pudiendo llegar a obtener un nuevo documento basado en la información del archivo original. Esto último se conseguirá gracias a las hojas de estilo XSLT y se realizará bajo encarga para cada uno de los casos.

Debido a que otro objetivo era que fuera un sistema altamente escalable, se optó por usar la tecnología "Cloud Computing" de Microsoft Azure de pago por uso. Esto permite contratar en base a lo que realmente se necesita. Además, nos brindaba herramientas como computación en la nube, almacenamiento de ficheros, alojamiento web, bases de datos, servicios de mensajería… consiguiendo finalmente todos los objetivos marcados.

Palabras clave: Integración, Escalabilidad, XSLT, Microsoft Azure

Abstract

The recent popularity of systems integration is indisputable. This assignment is focused on the integration of banking and finance sectors. The intended objectives were the simplification and improvement of communication systems, as well as the standardization and transformation of the information in an easily readable format for the recipient.

It has been raised a baseline scenery, consisted of banks and intermediaries. On this basis, banks are the part who share the information and the intermediaries the receivers. This project proposes a subscriptions system where each intermediary is capable, through a responsive web, of selecting all those banks from he wishes to receive information. In addition, the intermediary has the possibility of choosing the format of the obtained information (XML, JSON...) and of gaining a syntax and logic customized conversion, being able to have a new document based on the information of the original file. This process has been reached by XSLT style sheets and it is done on request of each particular case.

Since another goal was to develop a highly scalable system, the chosen option has been the pay-per-use technology of Microsoft Azure's Cloud Computing. This system enables the hiring of services based on the recipient's needs. Also, it offers some additional tools such as cloud computing, file storage, web hosting, databases and messaging service. Therefore, all stated objectives have been achieved.

Keywords: Integration, Scalability, XSLT, Microsoft Azure

Índice general

Índice Figuras

Capítulo 1

“Introducción”

1.1 Motivación

Cuando intentamos analizar las necesidades de una empresa, se puede comprobar claramente que una de ellas es la comunicación con otros servicios externos a ella. Por eso cuando hablamos de integración de servicios y gestión de la información se pueden observar 3 claros problemas:

a) Fragmentación de la información, es decir, demasiadas bases de datos repartidas por diferentes departamentos. Esto puede generar 2 situaciones:

   - Replicación de información suponiendo un coste adicional de mantenimiento y almacenamiento, además de posibles problemas de consistencia de la información.

   - Fragmentación de la información, afectando al rendimiento de un proceso que necesita tener una vista completa de un objeto que esta descompuesta en fragmentos ubicados físicamente en sitios distintos [1].

b) Falta de automatización de servicios. Y es que, desde hace unos años el aumento de sistemas que pueden conectarse a Internet y sobre todo con la aparición del Internet of Things, genera que sea una dura labor de realizar, sin ayuda de monitorización, y automatización de procesos repetitivos y metódicos.

c) Cada empresa utiliza una tecnología diferente. Esto provoca la existencia de una solución específica para cada comunicación.

Por ello, en este trabajo trataremos de diseñar e implementar una solución que permita disminuir el impacto de estos problemas. Concretamente se presentará una solución para un caso real, que es el del sector de las finanzas.

El sector de las finanzas se enfrenta a un problema de transferencia masivo de datos donde cada banco procesa su información en un formato determinado, y en gran

cantidad de veces, este formato es diferente al que un “intermediario” desea. Esto provoca que exista una cantidad excesivas de bases de datos, colas de mensajería y/o transformaciones de datos.

Concretamente en este sector no existe integración entre distintos bancos e intermediarios (Inmobiliarias, compañías de seguros…). Es decir, normalmente una “Entidad Bancaria X” tiene como propiedad o forma parte de un conglomerado de una “Compañía de Seguros X” y una “Inmobiliaria X” asociadas, entre las cuales, hay comunicación bajo un mismo nombre, por ejemplo, BBVA, pero no existe comunicación entre un segundo agente “Y” y sus respectivos asociados.

1.2 Contexto empresarial y requisitos iniciales

El proyecto sobre el que está basado este trabajo era un encargo para la empresa Everis, que es una compañía perteneciente a NTT Data, dedicada a la consultoría y outsourcing, abarcando sectores de telecomunicaciones, entidades financieras, industria, energía y recursos, seguros, administraciones públicas y banca. Everis llegó a facturar en el último ejercicio fiscal cerca de 1.173 millones de euros. Cuenta con 21.000 profesionales repartidos por Europa, USA y Latinoamérica.

Everis recibió un encargo para realizar una integración entre distintos bancos como eran ING, Santander, Caixa Bank y OpenBank, hasta un intermediario llamado SWIRL. Este intermediario realmente era otro equipo de trabajo de la compañía de Everis que pretendía realizar estadísticas, gráficas e informes con toda la información recibida y finalmente enviársela de vuelta a las respectivas entidades bancarias.

La finalidad de esta integración era convertir los ingentes archivos generados automáticamente y que emitían los bancos, para que llegaran a los sistemas de SWIRL mensajes muchos más reducidos y transformados lógicamente para una interpretación mucho más eficiente.

La razón por la que éramos nosotros quienes integraban lo servicios de los bancos y SWIRL, eran por cuestiones de agilizar los costes temporales, y, que SWIRL se encargase de realizar sus tareas rápidamente sin preocuparse del formato de entrada de los datos.

Estas transformaciones se realizaban de XML a JSON o incluso el mismo formato XML, pero modificando la sintaxis con conversores XSLT. Además, dado que el equipo era experto en tecnología Azure de Microsoft, todas las herramientas usadas eran las pertenecientes a Azure, tanto las colas, los servidores, bases de datos etc

Este proyecto no llegó a terminarse, porque así lo deseó el cliente final. Por tanto, todo el trabajo de fin de grado realizado es una adaptación de acuerdo a la envergadura de este tipo de trabajos. Además, cabe mencionar, que, debido a la estructura jerarquizada de la empresa, y mi rango en el equipo, no se me permitió tener acceso a la documentación técnica y explicativa necesaria para una correcta realización y descripción por mi cuenta.

1.3 Descripción general de la propuesta

La solución que este trabajo propone es crear un sistema de integración intermedio entre bancos e intermediarios, que procesará y distribuirá la información que éstos compartan.

El modo de funcionamiento estará planteado en modo suscripción, es decir, el intermediario podrá registrarse en nuestra web, y seleccionar aquellas entidades de las que desea obtener los datos.

Para ello, inicialmente se realizarán los convenios pertinentes con las entidades bancarias, que deseen obtener beneficios económicos y empresariales a la hora de compartir su información.

Por ello el objetivo era crear un sistema que consiguiera una reducción de colas de mensajería, y la reducción de métodos de conversión, además de realizar una copia de seguridad y un registro de todos los mensajes obtenidos, los mensajes enviados y procesos realizados por el sistema. Una vez cumplidos los objetivos anteriormente mencionados, podríamos hablar de un proyecto de integración convencional.

Por ello, otro objetivo era proporcionar al intermediario un servicio personalizado. La gran ventaja del sistema es que, no solo se permite al receptor elegir el formato en que desea recibir el archivo. Sino que permite al receptor obtener un servicio personalizado, y recibir el archivo con una lógica modificada, es decir, con los valores e

identificadores cambiados de forma personalizada para que sus sistemas interpreten la información de manera adecuada y eficaz. Todo esto se consigue mediante unas cuantas librerías y las hojas de estilo XSLT.

Es en esto último, los archivos XSLT, es donde radica uno de los mayores costes del proyecto, pues para cada una de las peticiones de conversión personalizada, es necesario crear una nueva hoja de estilo que puede ser completamente distinta para cada uno de los intermediarios. Cabe mencionar la complejidad de desarrollar estas hojas de estilo debido a la poca potencia del lenguaje y de las herramientas que proporciona, pues a la hora de realizar lógicas complejas, la tarea puede resultar imposible implementarla con las practicas convencionales. Se explicará más a fondo en el capítulo 6.3.

Por último, con este proyecto también se buscaba que fuese un sistema altamente escalable, y preparado para un posible crecimiento del número tanto de bancos, como de intermediarios y a su vez de la cantidad de información que se mueve por el sistema. Es por ello por lo que se ha optado por usar la tecnología de Microsoft Azure, que “es una nube pública de pago por uso que te permite compilar, implementar y administrar rápidamente aplicaciones en una red global de Data Centers (centros de datos) de Microsoft” [2], accesible al equipo técnico y usuarios desde cualquier lado.

Al ser pago por uso, nos permitirá realizar inicialmente una insignificante inversión de dinero, para el funcionamiento óptimo, y crear un sistema solido a pequeña escala. Y en base al crecimiento y la magnitud del proyecto simplemente se debería contratar mayor potencia de cálculo, de almacenamiento y de cualquier herramienta que así lo requiriese.

1.4 Objetivos

A modo de resumen del punto anterior, el objetivo general de este proyecto consiste en el desarrollo de un sistema de integración que permita la comunicación automática entre bancos e intermediarios. Para ello, se plantean los siguientes objetivos específicos:

- Diseño de una base de datos que permita tener un control de todo el sistema

- Creación de un sistema de mensajería que permita la comunicación entre bancos, intermediarios e incluso con los distintos subprocesos del sistema.
- Desarrollo de plantillas XSLT que permitan transformar tanto el formato de los archivos, como la lógica y la semántica de estos, bajo demanda de cada uno de los intermediarios.
- Creación de un subproceso que permita procesar los mensajes recibidos y realizar la conversión con sus respectivas plantillas XSLT.
- Creación de subprocesos que permitan automatizar el direccionamiento de los mensajes.
- Desarrollo de una página web que permita, por un lado, realizar las gestiones de administrador al equipo técnico y, por otro lado, permitirle al intermediario que sea él mismo quien gestione los datos que desea recibir.

1.5 Estructura de la memoria

A lo largo de los capítulos se irán viendo en profundidad los diferentes aspectos que se han visto involucrados y que se han trabajado para el desarrollo de la solución de integración entre bancos e intermediarios.

En el capítulo 2 se explicarán las bases teóricas de estas tecnologías, el Background y tecnología ya existente que realizan la misma función que este sistema. El capítulo 3 será una continuación del 2, pues nos apoyaremos en algunos conceptos explicados previamente, para entender correctamente las herramientas usadas en el proyecto. La metodología usada para desarrollar el proyecto se explicará en el capítulo 4.

El capítulo 5 explicará en detalle la arquitectura del sistema y el modo de funcionamiento de forma detalla, y dividida en 3 fases perfectamente marcadas. El capítulo 6 será dedicado íntegramente al tema de las transformaciones, tanto de formato como de sintaxis. El capítulo 7 explicará la finalidad de la web y como está implementad. Finalmente, el capítulo 8 serán las conclusiones finales tras haber desarrollado el proyecto.

Capítulo 2

“Background”

Dado que este trabajo tiene como principal objetivo conectar dos servicios externos y realizar una integración de estos, a la par que una conversión, explicaré que tecnologías ya existían para cubrir estas necesidades.

Partiendo de que no existe ningún caso real el cual pueda ser consultado, pues puede que el proyecto no sea de índole público, intentaré hacer un Background de Integración de servicios, de transformación de archivos, y de las técnicas usadas actualmente.

Desde el punto de vista de la integración, actualmente podemos distinguir integración en 3 distintos aspectos.

Integración de Datos

Transferencia de Ficheros

- La forma más simple de integración
- Adecuada cuando las actualizaciones no son habituales
- Requiere consensos/estándares para el intercambio

Bases de Datos Compartidas

- Intercambio rápido de datos
- Se imponen fuertes requisitos de cooperación
- Modelo más rígido.
- Extensibilidad técnica viable, aunque a la hora de una implementación ‘real’ puede llegar a ser bastante complejo de realizar.

Invocación de Procedimientos Remotos

- Se aplica el principio de encapsulación escondiendo los datos para mantener su integridad
- Requieren publicar y conocer interfaces de procedimientos
- Puede imponer restricciones de alcance y visibilidad, según la tecnología empleada
- Se pueden desarrollar como ‘capas’ sobre SW ya existentes
- Los Servicios Web proporcionan estándares abiertos, como SOAP

Mensajería

- Alta velocidad
- Comunicación Asíncrona
- Confiable
- Permite desarrollar arquitecturas desacopladas
- Necesidad de consensos en mensajes
- Fácilmente extensible

Una vez conocemos estas estrategias podemos introducir el concepto Enterprise Application Integration (EAI) (Figura 1). Un EAI se define como el uso de software y principios arquitectónicos (de software) para resolver los problemas complejos de integración en aplicaciones empresariales, usando las distintas estrategias anteriormente mencionadas [3]. Un EAI nos permite:

- Compartir información entre aplicaciones (básicamente conectar diferentes Bases de Datos) y mantener los datos consistentes.
- Reducir potencialmente el panorama tecnológico, reduciendo la heterogeneidad, aplicando soluciones y tecnologías estándares, que a su vez se basan sobre otros estándares
- Agilizar y facilitar el despliegue de una aplicación nueva o una actualización de una existente. Esto se consigue definiendo estándares para la integración, lo que nos evitarán tener que modificar las tecnologías middleware

Un último concepto importante antes de contextualizar nuestro proyecto es el de ESB (Enterprise Service Bus), que es un modelo de arquitectura de software que gestiona la comunicación entre servicios web.

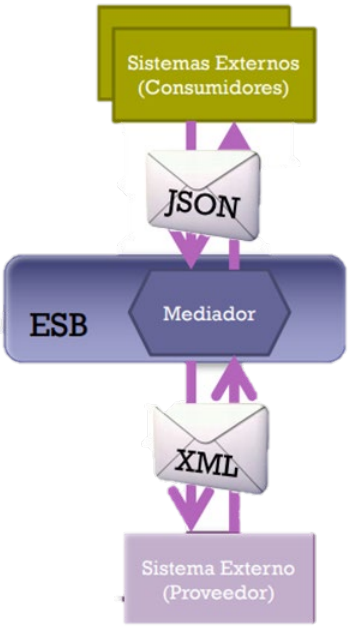


Figura 1: EAI con conversión de formato

De esta manera con un simple vistazo, podemos ver cuál ha sido la evolución respecto de los tradicionales, a los de la actualidad, siendo este último el modelo que se ha seguido para este proyecto (Figura 2).

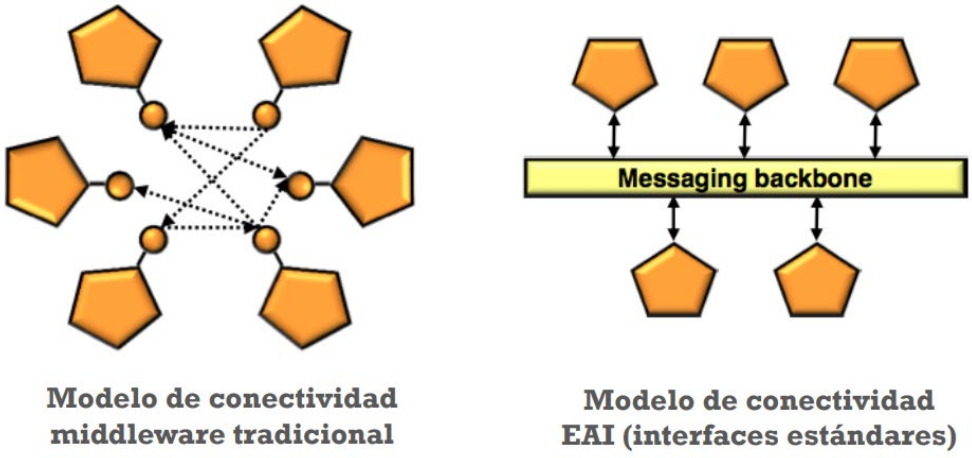


Figura 2: Diferencia entre Modelo tradicional y modelo actual. Extraída de [3]

Por otro lado, y dejando a un lado el transporte y la arquitectura, no centramos la transformación de datos. Para nuestro sistema hemos utilizado las hojas de estilo XSLT, pero dado que se le dedicará el capítulo 7 íntegramente a esto, ahora solo se explicarán algunas alternativas.

Podemos intentar usar **librerías predefinidas** específicas (Figura 3).

Formato Origen	Formato Destino	Transformador
EDI	XML	EDIReader
XML	*	XSLT / XPath xml2csv-conv
JSON	CSV	JSFiddle
…	…	…

Figura 3: Librerías de conversión ya existentes

Por otro lado, si no existen librerías específicas, se pueden usar lenguajes de manipulación, que nos permitirán crear una transformación usando lenguajes de lectura (parsers) de formatos(Figura 4) [4].

Formato	Lenguajes/Parsers
XML	SAX, DOM JAXP
BD	SQL
CSV	openCSV csvReader
…	…

Figura 4: Formatos que puede modificar cada Lenguaje/Parser

Y por último podemos utilizar herramientas de transformación de datos, que nos permiten “mapear” desde un entorno gráfico amigable. Además, nos ofrece operadores para recorrer y manipular las estructuras definidas, abstrayéndonos del lenguaje de transformación.

- Mule Data Mapper
- BizTalk Server Data Mapper
- IBM WebSphere sMash (Figura 5)
- Altova MapForce
- Talend Data Transformation

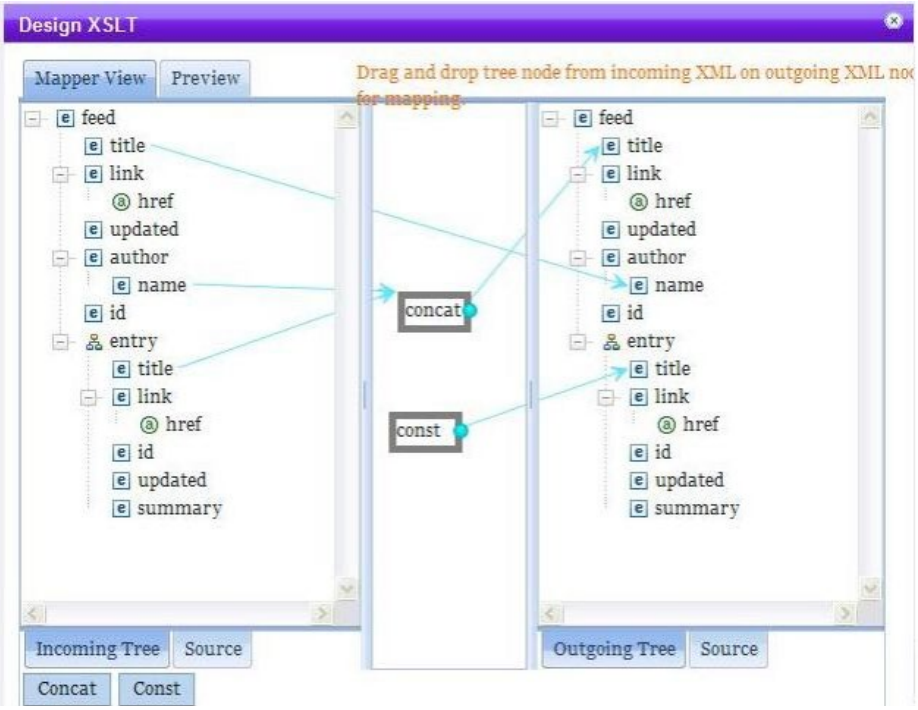


Figura 5: Captura de pantalla IBM WebSphere sMash

Por último, y con el fin de entender algunas de las tecnologías usadas para nuestro proyecto, es importante exponer como alojar código en la nube para que este pueda manejar las colas, y transformadores.

Tradicionalmente, hemos creado e implementado aplicaciones web en las que tenemos cierto grado de control sobre las solicitudes HTTP que se realizan a nuestro servidor. Nuestra aplicación se ejecuta en ese servidor y somos responsables de aprovisionar y administrar los recursos para ella. Hay algunos problemas con esto.

1. Se nos cobra por mantener el servidor activo incluso cuando no estamos atendiendo ninguna solicitud.
2. Somos responsables del tiempo de actividad y mantenimiento del servidor y todos sus recursos.
3. También somos responsables de aplicar las actualizaciones de seguridad apropiadas al servidor.
4. A medida que el uso aumenta, necesitamos administrar la escala hacia arriba de nuestro servidor. Así mismo, administrar la escala hacia abajo cuando no tengamos tanto uso.

Para pequeñas empresas y desarrolladores individuales, esto puede suponer mucho trabajo. Esto termina distrayéndonos del trabajo más importante que tenemos; construir y mantener la aplicación actual. En las organizaciones más grandes, esto lo maneja el equipo de infraestructura y, por lo general, no es responsabilidad del

desarrollador individual. Sin embargo, los procesos necesarios para apoyar esto pueden terminar ralentizando los tiempos de desarrollo. Ya que no se puede seguir adelante en la construcción de la aplicación sin trabajar con el equipo de infraestructura para poder ponerse en marcha [5].

Como alternativa a lo anteriormente mencionado, y es la estrategia escogida en nuestro proyecto, existe la computación sin servidor. Nosotros concretamente hemos escogido Microsoft Azure, que explicaremos en profundidad en el capítulo 3.

Capítulo 3

“Contexto tecnológico”

Una de las bases del proyecto era que fuera fácilmente escalable y que todo estuviera en funcionamiento 24 horas al día durante todo el año. Esto para un proyecto pequeño supondría tener que comprar o contratar servidores. Pero esto supone un problema, ya que la cantidad de datos y potencia de calculo que necesita el sistema es variable, y obtener una gran capacidad computacional que luego no fuera utilizada podría suponer perdidas. Y a la inversa, una carga computación insuficiente, podría provocar caídas del sistema, perdidas de información e incluso pérdidas económicas por insatisfacción de los clientes.

Por tanto, la solución por la que se ha optado es por realizar una contratación de pago por uso, es decir, pagar cada mensaje encolado en el sistema, por cada proceso realizado en la nube, por cada MB ocupado en la base de datos. Así que optamos por elegir la tecnología de Microsoft Azure que nos brindaba todas estas herramientas con una gran integración entre sus propios componentes.

Además, y aunque se explicará más tarde en el Capítulo 5, sobre arquitectura, es importante mencionar el tipo de estrategia de comunicación usada para la comunicación de nuestra arquitectura, pues ayudará a entender porque se usan algunas herramientas. Se trata de una integración de **servicios basada en Mensajería:**

- Alta velocidad
- Comunicación Asíncrona
- Confiable
- Permite desarrollar arquitecturas desacopladas
- Necesidad de consensos en mensajes
- Fácilmente extensible

Además, tal y como se expone en la asignatura de Integración de Aplicaciones, cuando se busca una integración a gran escala, se busca desarrollar Middlewares ligeros, y el uso de Buses Software Empresariales (EAI), y esa ha sido la base de este proyecto.

3.1 Azure Functions

Antes de explicar que son las Functions de Azure es importante conocer el concepto **Serverless.** Como ya he explicado anteriormente, en el pasado se pagaba por contratar potencia computacional y si no se usaba o faltaba, suponía un problema, así que esto supone una solución.

La computación sin servidor (o serverless para abreviar) es un modelo de ejecución en el que el proveedor en la nube (en nuestro caso Microsoft Azure) es responsable de ejecutar un fragmento de código mediante la asignación dinámica de los recursos. Y cobrando solo por la cantidad de recursos utilizados para ejecutar el código. El código, generalmente, se ejecuta dentro de contenedores sin estado que pueden ser activados por una variedad de eventos que incluyen solicitudes HTTP, eventos de base de datos, servicios de colas, alertas de monitoreo, carga de archivos, eventos programados (trabajos cron), etc. El código que se envía a al proveedor en la nube para la ejecución es generalmente en forma de una función. Por lo tanto, serverless a veces se denomina “Funciones como servicio” o “FaaS” [5]. Por tanto, una Azure Function se puede resumir como la unión de código en la nube, disparado por un evento con capacidad para manejar información (Figura 6).

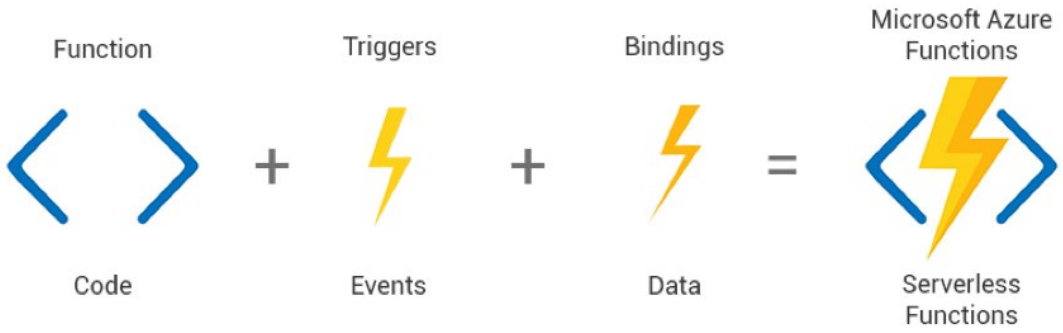


Figura 6: Explicación del logo de Microsoft Azure Functions

3.2 Azure Service Bus

Lo primero es mencionar que este servicio de mensajería sigue una arquitectura ESB. Un Bus de Servicio Empresarial (ESB por sus siglas en inglés), (aunque ya explicado en el Capítulo 2), es un modelo de arquitectura de software que gestiona la

comunicación entre servicios web. Es un componente fundamental de la Arquitectura Orientada a Servicios [6] [7].

Azure Service Bus es un servicio de mensajería multiinquilino (multiinquilino: capacidad de administrar y crear particiones de datos de sitios y de otros servicios o software compartidos con el fin de dar cabida a varios inquilinos) en la nube que puede usarse para enviar información entre aplicaciones y servicios. Las operaciones asincrónicas le ofrecen una mensajería flexible y asincrónica, además de funcionalidades de mensajería estructurada de tipo FIFO (primero en entrar, primero en salir) y de publicación-suscripción, es decir crear Colas y Topics.

El coste de este servicio dependerá de las características que deseamos obtener, aunque el plan básico parte de €0,043 por millón de operaciones [8].

3.3 Azure SQL Server y DataBase

Para el correcto entendimiento de estas dos tecnologías hay que pensar en las clásicas tecnologías de bases de datos donde inicialmente se crea la base de datos y esta está alojada en un servidor. Pues en este caso Azure sigue la misma arquitectura, donde Azure SQL Server se encarga de gestionar el alojamiento, el acceso, seguridad, permisos… de todas las bases de datos de SQL DataBase que deseamos.

Otras grandes ventajas de este servicio son, por un lado, la fácil integración con .NET para realizar la web de este proyecto, sin necesidad de lenguajes como PHP, y, por otro lado, que cuenta con una potente herramienta de escritorio, SQL Server, que es más que un RDBMS (Sistema de gestión de bases de datos relacionales), ya que este, a diferencia de otros servicios como MySQL, cuenta con el gran respaldo de Microsoft. Esto significa gran cantidad de herramientas ya integradas en la aplicación y listas para el funcionamiento, ya que con otras tecnologías se deben buscar herramientas de terceros [9].

3.4 Azure Blob Storage

Azure Blob Storage es la solución de almacenamiento de objetos de Microsoft para la nube. Blob Storage está optimizado para el almacenamiento de cantidades masivas de datos no estructurados. Los datos no estructurados son datos que no cumplen un modelo de datos o definición concreta, como texto o datos binarios.

Los usuarios o las aplicaciones cliente pueden acceder a objetos en Blob Storage a través de HTTP/HTTPS, desde cualquier lugar del mundo. Se puede acceder a los objetos de Blob Storage mediante la API REST de Azure Storage, Azure PowerShell, la CLI de Azure o una biblioteca de cliente de Azure Storage. Las bibliotecas de cliente están disponibles para numerosos lenguajes, como .NET, Java, Node.js, Python, Go, PHP y Ruby.

3.5 XML

XML, siglas en inglés de eXtensible Markup Language, traducido como "Lenguaje de Marcas Extensible", es un metalenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible. Proviene del lenguaje SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML) para estructurar documentos grandes. A diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciones deben comunicarse entre sí o integrar información [10].

XML no ha nacido únicamente para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Tiene un papel muy importante en la actualidad ya que permite, con la ayuda de otras herramientas, la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil [11].

3.6 JSON

JSON (acrónimo de JavaScript Object Notation, «notación de objeto de JavaScript») es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera (año 2019) un formato independiente del lenguaje [12].

En la actualidad se considera a pequeños rasgos a JSON un poco más ligero que XML, pero en la práctica, los argumentos a favor de su rendimiento son poco relevantes. Debido a pequeñas cuestiones de seguridad y el auge del procesamiento nativo de XML incorporado en los navegadores modernos, XML y JSON se usan a la par, incluso coexistiendo ambos formatos en un mismo sistema [12]. Por esa razón, JSON se emplea habitualmente en entornos donde el tamaño del flujo de datos entre cliente y servidor es de vital importancia (de aquí su uso por Yahoo!, Google, Mozilla, etc., que atienden a millones de usuarios) cuando la fuente de datos es explícitamente de fiar y donde no es importante el hecho de no disponer de procesamiento XSLT para manipular los datos en el cliente.

3.7 XSLT

XSLT o Transformaciones XSL es un estándar de la organización W3C que presenta una forma de transformar documentos XML en otros e incluso a formatos que no son XML.

Las hojas de estilo XSLT, aunque el término de hojas de estilo no se aplica sobre la función directa del XSLT, realizan la transformación del documento utilizando una o varias reglas de plantilla. Estas reglas de plantilla, unidas al documento fuente a transformar, alimentan un procesador de XSLT, el que realiza las transformaciones deseadas poniendo el resultado en un archivo de salida [13].

Actualmente, XSLT es muy usado en la edición web, generando páginas HTML o XHTML. La unión de XML y XSLT permite separar contenido y presentación, aumentando así la productividad.

Capítulo 4

“Metodología”

A la hora escoger una metodología de trabajo para este trabajo fue muy clara la elección, metodología incremental, y a continuación se entenderá el porqué, y que es esto.

El modelo incremental de gestión de proyectos tiene como objetivo un crecimiento progresivo de la funcionalidad. Es decir, el proyecto va evolucionando con cada una de las entregas previstas hasta que se amolda a lo requerido por el cliente o destinatario [14].

En cada una de las entregas o fases, el proyecto debe mostrar una evolución con respecto a la fecha anterior; nunca puede ser igual.

Una de las claves para que esto se haga efectivo es la evaluación de las etapas. Se deben analizar si los resultados parciales son los esperados y si, sobre todo, apuntan al objetivo principal. De no ser así, se deberá intervenir en él e implementar las soluciones que la situación requiera.

El modelo de gestión incremental no es un modelo necesariamente rígido, es decir, que puede adaptarse a las características de cualquier tipo de proyecto:

1. **Comunicación con el cliente y objetivos**: son los objetivos centrales y específicos que persigue el proyecto, según la demanda del cliente.
2. **Planeación, definición de las tareas y las iteraciones**: teniendo en cuenta lo que se busca, el siguiente paso es hacer una lista de tareas y agruparlas en las iteraciones que tendrá el proyecto. Esta agrupación no puede ser aleatoria. Cada una debe perseguir objetivos específicos que la definan como tal.
3. **Diseño de los incrementos**: establecidas las iteraciones, es preciso definir cuál será la evolución del producto en cada una de ellas. Cada iteración debe superar a la que le ha precedido. Esto es lo que se denomina incremento.
4. **Desarrollo del incremento**: posteriormente se realizan las tareas previstas y se desarrollan los incrementos establecidos en la etapa anterior.

5. **Validación de incrementos**: al término de cada iteración, los responsables de la gestión del proyecto deben dar por buenos los incrementos que cada una de ellas ha arrojado. Si no son los esperados o si ha habido algún retroceso, es necesario volver la vista atrás y buscar las causas de ello.
6. **Integración de incrementos**: una vez son validados, los incrementos dan forma a lo que se denomina línea incremental o evolución del proyecto en su conjunto. Cada incremento ha contribuido al resultado final.
7. **Entrega del producto**: cuando el producto en su conjunto ha sido validado y se confirma su correspondencia con los objetivos iniciales, se procede a su entrega final.

A continuación, en la Figura 7, se plasman las fases anteriormente mencionadas.

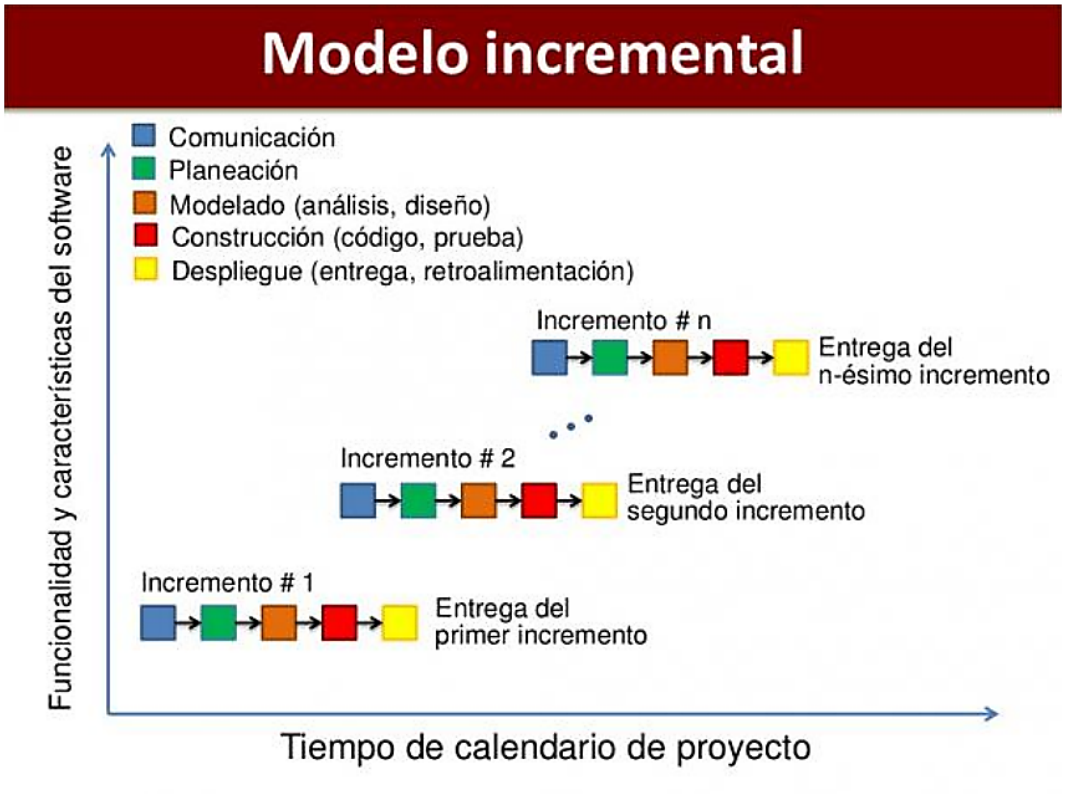


Figura 7: Ejemplo de planificación con Modelo Incremental. Extraído de [15]

Concretamente este proyecto estaba dividido en 3 grandes fases muy bien definidas, que posteriormente en el capítulo 5, se explicarán más a fondo. A su vez cada una de estas grandes fases estaba divida en pequeñas tareas que seguían el mismo modelo. De este modo, vamos a mostrar todas las fases y cada uno de los

incrementos, sabiendo que para cada uno de los puntos, ser realizaban todos los pasos anteriormente mencionado de “Comunicación, Planeación, Construcción…”.

Fase 1

- Inserción y obtención de información de cada uno de los bancos en Base de Datos
- Recepción y procesamiento de los mensajes de cada uno de los bancos en base a la información registrada en la Base de Datos
- Conexión y descarga de archivos del servicio de almacenamiento Blob Storage

Validación de todas las subfases trabajando juntas

Fase 2

- Ejecución de varias consultas en cadena a Base de Datos, para conseguir toda la información necesaria para la transformación de cada uno de los mensajes, y asignación en las variables locales pertinentes.
- Creación del proceso de transformación estándar sin plantilla XSLT
- Creación del proceso de transformación mediante plantilla XSLT

Validación de todas las subfases trabajando juntas y a su vez con la fase 1

Fase 3

- Obtención de Base de Datos de la información necesaria para el envío de ese mensaje a la cola necesaria
- Conexión y carga del archivo resultante al servicio de almacenamiento Blob Storage del Intermediario
- Conexión a la cola necesaria y envío del mensaje

Validación de todas las subfases trabajando juntas, y a su vez validación global del sistema.

Por otro lado, me parece relevante mencionar el **Método Kaizen**, y es que, aunque no sea una metodología de trabajo, sino más bien una filosofía o ámbito de buenas prácticas, se ha utilizado como referencia para gestionar la realización del proyecto inicialmente durante mi trabajo en la empresa Everis, con quien inicié este proyecto.

Con este método se buscan mejoras en la productividad, la eficacia, la seguridad y la reducción de residuos. Pero sus beneficios se extienden mucho más allá, pues, aunque la mayoría de los cambios pueden no resultar ser de grandes dimensiones, su impacto sí es susceptible de impulsar repercusiones significativas en un futuro medio [16].

El método a seguir:

1. Establecimiento de metas claras y realistas, bien documentadas.
2. Revisión del estado actual de la situación y desarrollo de un plan de optimización.
3. Implementación de mejoras.
4. Revisión y aplicación de las correcciones necesarias.
5. Elaboración de un informe de resultados y determinar los elementos de seguimiento.

Las ventajas de aplicar el método Kaizen no se limitan a un aumento de la productividad, sino que se trasladan a otros ámbitos, contribuyendo a lograr[16] :

- Disminución de la generación de residuos: al ganar en eficiencia y utilizar mejor las habilidades de los empleados se minimizan los desechos, todos esos elementos que no producen valor.
- Aumento de los niveles de satisfacción: un hecho que tiene un impacto directo en la forma en que se hacen las cosas, iniciando un ciclo de motivación que se mantiene en el tiempo.
- Mayor grado de compromiso: los miembros del equipo presentan un mayor interés en su trabajo y son más proclives a comprometerse con las metas de la organización.
- Mejores tasas de retención del talento: cuando las personas se encuentran satisfechas y motivadas son más propensas a quedarse, ya que no necesitan buscar en otros lugares lo que ya han conseguido y les depara un futuro prometedor.
- Incremento de la competitividad: el aumento de la eficiencia contribuye a lograr costos más bajos y productos de mejor calidad, mejorando el posicionamiento de la empresa en el mercado.
- Impulso a los niveles de satisfacción de los consumidores: que obtienen un mejor servicio y se benefician de productos de mayor calidad y con menos defectos.

- Optimización de la resolución de problemas: al enfocar los procesos desde una perspectiva de búsqueda de soluciones, los propios empleados están capacitados para resolver problemas de forma continua.
- Fortalecimiento de los equipos: al trabajar juntos para resolver problemas, gracias al método Kaizen se fortalecen los vínculos y se construyen equipos mejores y más resistentes, preparados para afrontar cualquier desafío.

Capítulo 5

“Arquitectura”

5.1 Visión general del sistema

Antes de entrar a explicar cada uno de los componentes del sistema vamos a ver el sistema de manera más simplificada.

Para una mejor explicación dividiremos todo el proceso en tres fases claramente distinguidas. En la primera fase se explica cómo obtiene nuestro sistema toda la información de cada uno de los bancos, los problemas a los que nos hemos enfrentado y como solucionarlos. La segunda fase explica como procesar esa información, y la fase tres describe como realizar el reparto de información a cada uno de los intermediarios. En la Figura 8 se puede observar un esquema de las distintas fases en orden cronológico

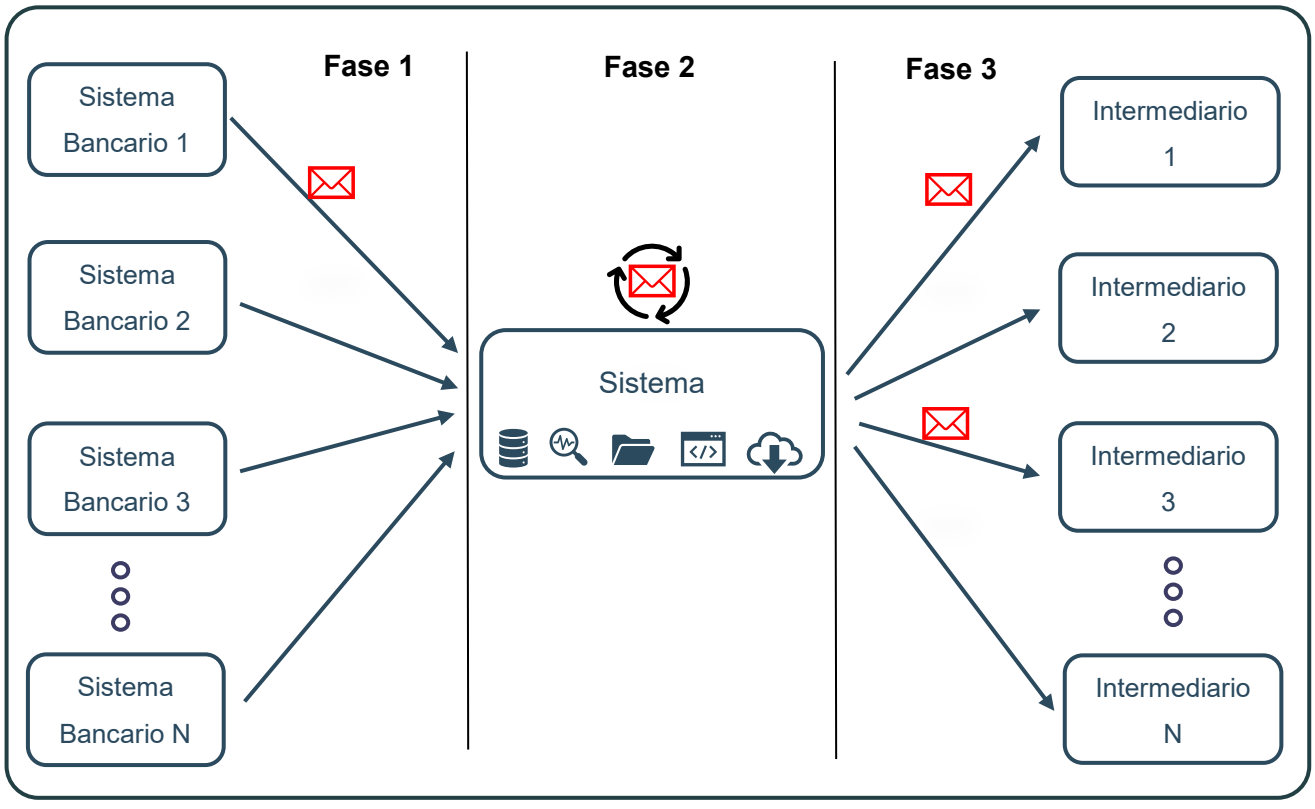


Figura 8: Visión general del funcionamiento del sistema divido en 3 fases

También es importante hacernos una visión general de la estructura de la base de datos visible en la Figura 9, que nos servirá para entender gran cantidad de cosas más adelante.

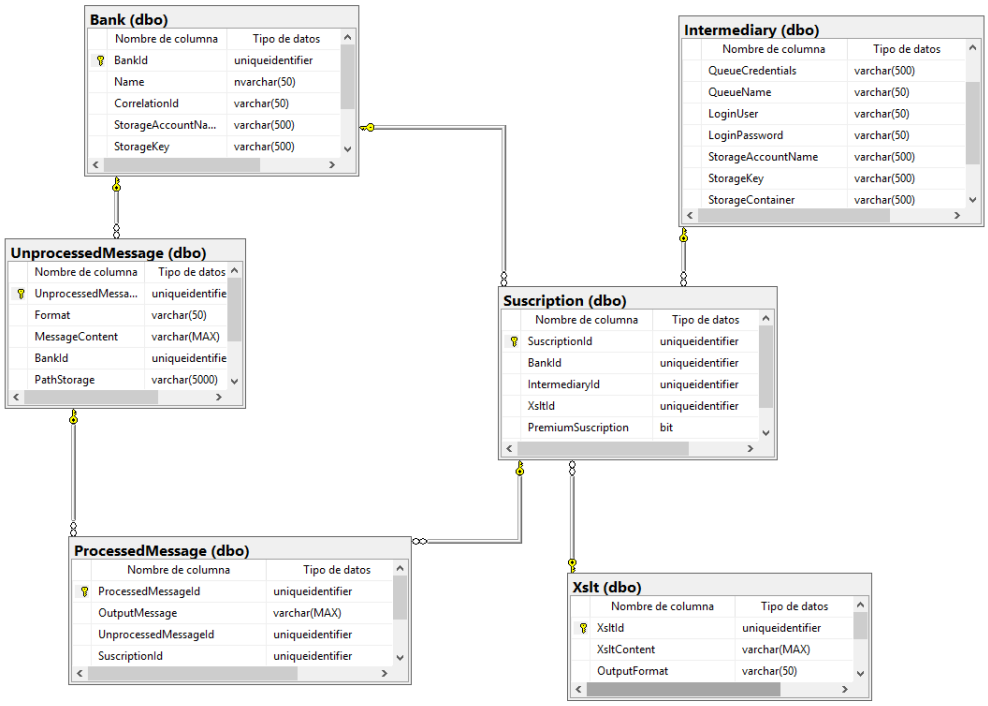


Figura 9: Esquema global de la Base de Datos

A lo largo de las distintas fases, iremos explicando la base de datos poco a poco, según se vayan necesitando las distintas tablas. Pero a modo resumen, la tabla “Suscription” es una tabla relacional entre un banco y un intermediario, de modo que “un” banco pueda tener “muchas” suscripciones. Por otro lado, una suscripción tiene asociada una tabla Xslt con su conversión respectiva, un Intermediario, y el mensaje transformado y sin transformar (Tablas ProcessedMessage y UnprocessedMessage).

Finalmente, es importante mencionar otro componente importante del sistema, la web, pues es la cara externa del sistema. La web, permitirá 2 cosas muy importantes.

Por un lado, permitirá al administrador del sistema dar de alta a nuevos bancos que comenzarán a enviar información a nuestro sistema. Además de realizar todas las tareas necesarias relacionadas con la base de datos, como es modificar usuarios, suscripciones, hojas de estilo, bancos…Por otro lado, permitirá a los intermediarios registrarse en nuestro sistema, y comenzar a obtener información de los distintos bancos que existen en nuestro sistema.

5.2 Fase 1: Recepción de datos en el sistema

Como hemos mencionado antes, mediante la web, se realizará un registro en base de datos, de cada uno de los bancos que desean obtener beneficios empresariales al compartir esta información.

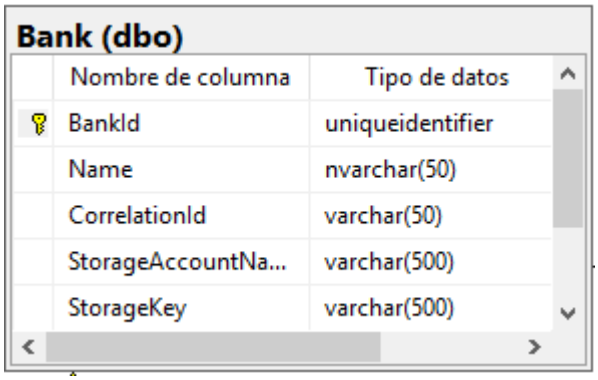


Figura 10: Tabla Bank

Estos bancos se registrarán en la base de datos obteniendo de ellos información básica visible en la Figura 10 como, las cadenas de conexión necesarias para conectarse al sistema de mensajería, y las cadenas de conexión para conectarse al sistema de almacenamiento. Es importante explicar esto último de sistemas de almacenamiento.

Los archivos originales que enviaban los bancos ocupaban gran capacidad debido a las ingentes cantidades de información, y debido a las limitaciones técnicas del Service Bus (el sistema de mensajería), los mensajes que se enviaban podían llegar incompletos. Así que se optó por la siguiente estrategia.

Inicialmente se creará un sistema de almacenamiento en la nube, donde cada banco automáticamente subirá su archivo en sus cuentas de almacenamiento, respectivamente configuradas para recibir peticiones REST y que nuestro sistema obtenga el archivo.

Una vez que los respectivos bancos han subido el archivo a la nube, el banco ya no envía el contenido original del archivo, sino un mensaje con la ubicación/referencia donde se encuentra el archivo.

Así que, en el momento que recibimos un mensaje de un banco determinado, solo necesitamos conocer las credenciales del Blob Storage del banco respectivo, para descargar el archivo que venía referenciado en el mensaje.

Antes de continuar, mencionar la facilidad de las herramientas de Azure, y concretamente del Blob Storage, pues en un simple menú de administración tenemos toda las herramientas e información necesaria, de manera bastante visual como se puede apreciar en la Figura 11.

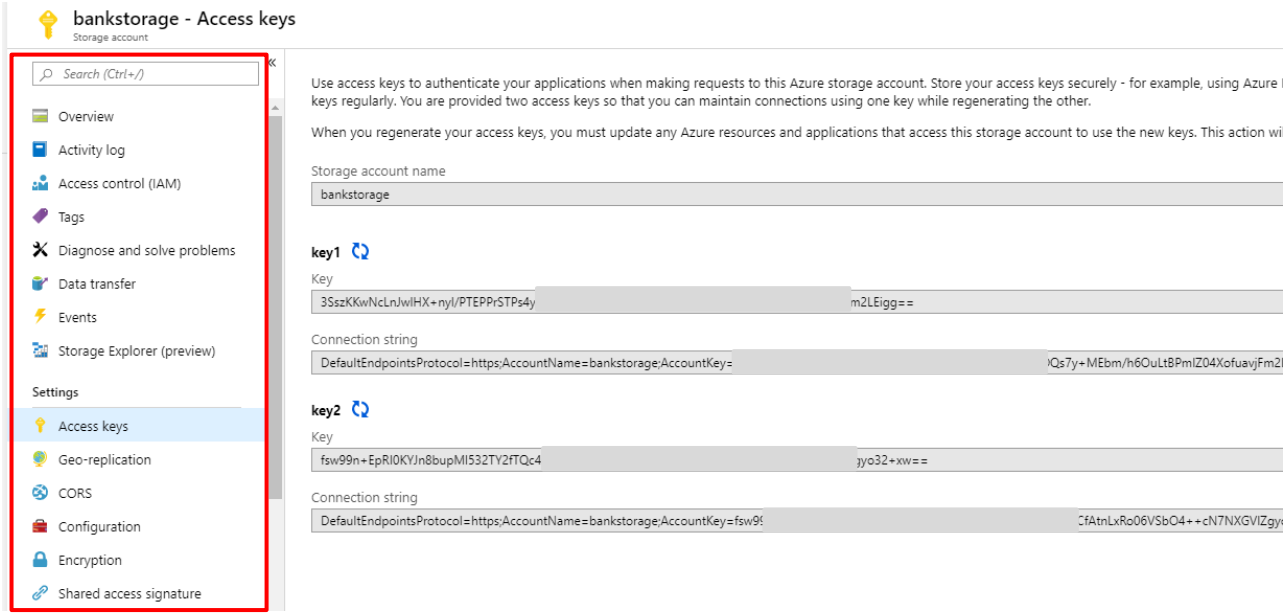


Figura 11: Portal Azure con todas las herramientas administrativas del Blob Storage

Como hemos dicho anteriormente, no se enviará el archivo original mediante servicios de colas. Mediante estas colas, solamente se enviará un mensaje en formato JSON con información sobre donde está alojado el archivo original. Además, como ya tenemos todas las credenciales de la entidad bancaría en la base de datos, no será necesario que el banco las envié con cada mensaje, consiguiendo así, asegurar la privacidad de esa información.

Así que los bancos enviaran un mensaje con el formato de la Figura 12:

```
{
   "inputFormat": "xml",
   "fileName": "asdi3lll3-ll33dfl-ladf-laj23-2-22",
    "container": "containerBankX"
}
```

Figura 12: Formato estándar de mensaje usado por los bancos

Este es el formato que se ha establecido como convenio para todos los bancos, y todos ellos enviarán los mensajes en este formato a la puerta de entrada de nuestro sistema. La puerta de entrada de nuestro sistema es una cola del Service Bus de Azure. Es importante mencionar que las colas siguen el método FIFO (First Input, First Out) y además de manera asíncrona.

Ahora nos encontramos con una problemática, en la Figura 12, como hemos mencionado anteriormente, los mensajes que recibimos contienen el nombre del archivo y la carpeta donde está, pero no sabemos cuál es la cuenta de almacenamiento ni las credenciales de acceso (por seguridad de esa información). Además nos llegan mensajes de distintos bancos, es decir, que es imposible localizar el archivo solo con el contenido del mensaje. Esto se resuelve con una propiedad del Service Bus, llamada CorrelationId, que actuará como el conocido "remitente", es decir, cuando un banco envía un mensaje, este siempre irá marcado con el CorrelationId de cada banco. Con esta referencia solo tenemos que buscar en BD a quien corresponde ese CorrelationId y buscar cuál es su cuenta de almacenamiento y sus credenciales, como se puede ver en la Figura 14.

BankId	Name	CorrelationId	StorageAccountName	StorageKey
D9B98B65-3501-4459-B496-07E95FC9A9D7	Santander	112SA561	bankstorage	3SszKKwNcLr

Figura 13: Un registro de la tabla Bank

Así que finalmente el proceso realizado en la primera fase es el mostrado en la Figura 14:

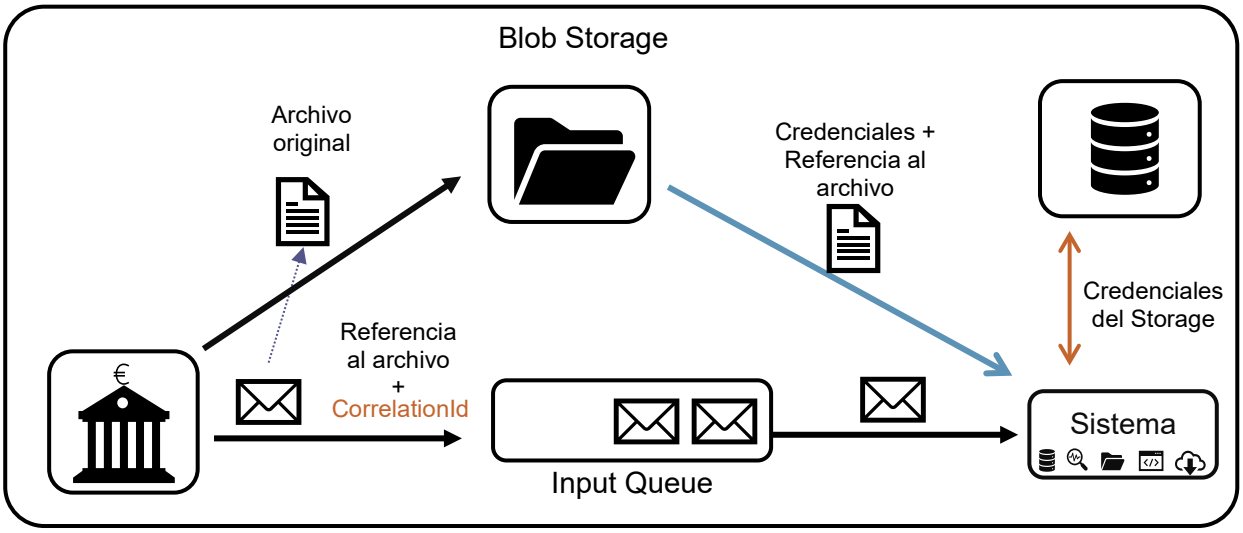


Figura 14: Proceso realizado en la Fase 1

En la esquina inferior derecha de la Figura 14 se puede observar que hay un rectángulo con el texto “Sistema”. Esto es una abstracción de lo que es realmente nuestro sistema.

Realmente el encargado de realizar la lógica anteriormente explicada (consultar en la base de datos y descargar archivos del Blob Storage), es una Azure Function, que se disparará automáticamente cuando se detecte que hay mensajes sin procesar en la cola “Input” del Service Bus.

Internamente nuestro sistema está construido tal y como se muestra en la Figura 15, que a continuación será explicada:

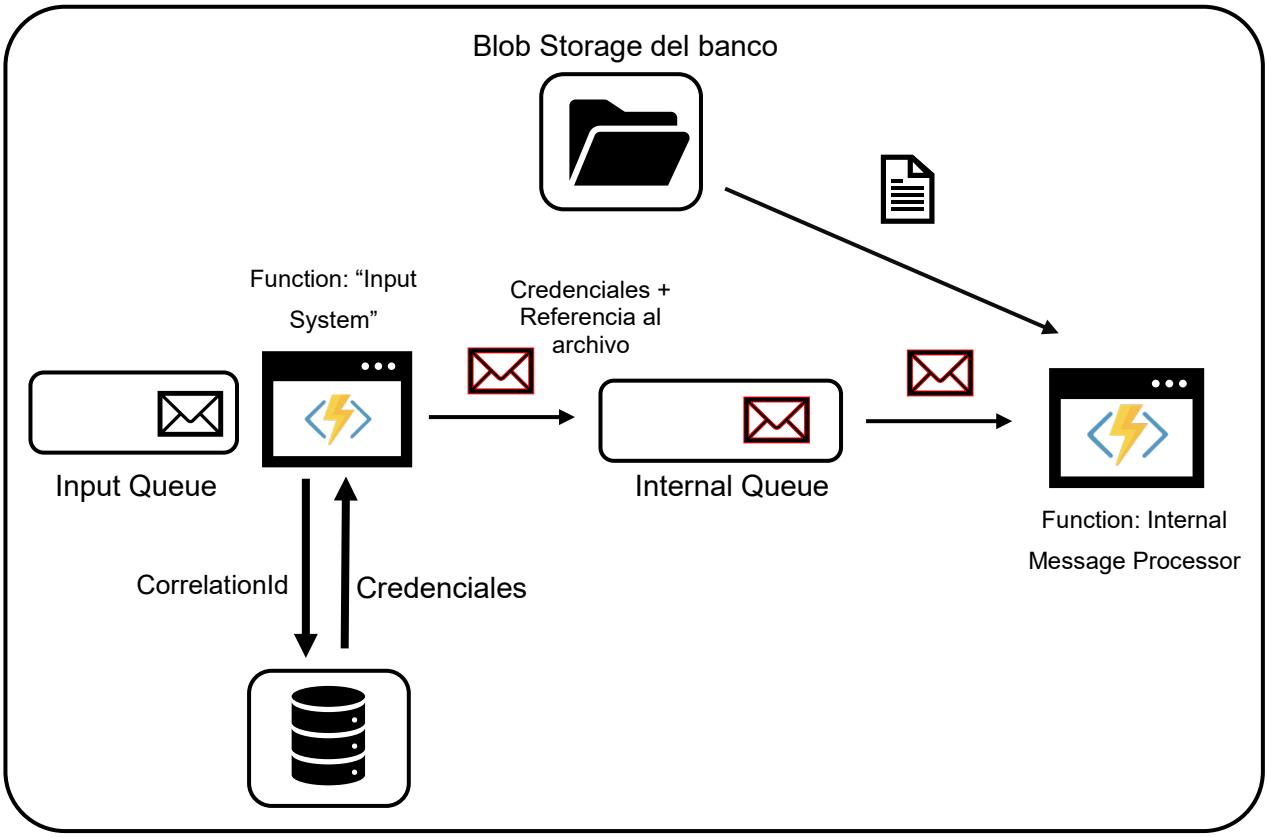


Figura 15: Funcionamiento interno del sistema en la Fase 1

Nuestro sistema tiene una primera Function llamada “Input System” que se encarga de conseguir recibir todos los mensajes que están en la cola exterior al sistema. Esta función se ejecutará una vez por cada mensaje recibido y solo se encargará de obtener las credenciales e información necesaria para que posteriormente se descargue el archivo.

Una vez recopilada se ha recopilado toda la información necesaria sobre el archivo, la Function Input System enviará está información a la Function “Internal Message Processor” mediante otro Service Bus que hará las veces de comunicador interno. El formato del mensaje que se inserta en esta nueva cola será JSON y con la estructura de la Figura 16:

```
{
"bankId": "d9b98b65-3501-4459-b496-07e95fc9a9d7",
"bankName": "BANCO X",
"inputFormat": "xml",
"fileName": "FILENAME",
"storage":
    {
    "accountName": "bankstorage",
    "key": "3SszKKwNcLnJwlHX+nyl/PTEPPrSTPs4yP814XpZcffBXDzDQs7y+l
    "container": "xmlmessages"
    }
}
```


Figura 16: Formato mensajes internos

Una vez se inserta el mensaje en la cola “Internal Queue”, la Function “Processor” se dispara automáticamente, y descarga el archivo que previamente había subido la entidad bancaria, pudiendo decir que ya hemos recibido el archivo original en nuestro sistema.

5.3 Fase 2: Suscripciones y transformaciones

Para poder explicar esta segunda fase es necesario realizar unas contextualizaciones previas.

Como hemos mencionado al inicio del Capítulo 5, inicialmente los intermediarios se darán de alta en la base de datos junto con toda la información necesaria. Entonces se creará, lo que para nuestro proyecto se ha denominado “suscripción”. Cuando un intermediario desea recibir datos de un banco concreto, este entrará en la web y creará una “suscripción” a los mensajes de este banco. Técnicamente lo que se hace es crear una tabla relacional entre un Banco y un Intermediario como se puede ver en la Figura 17

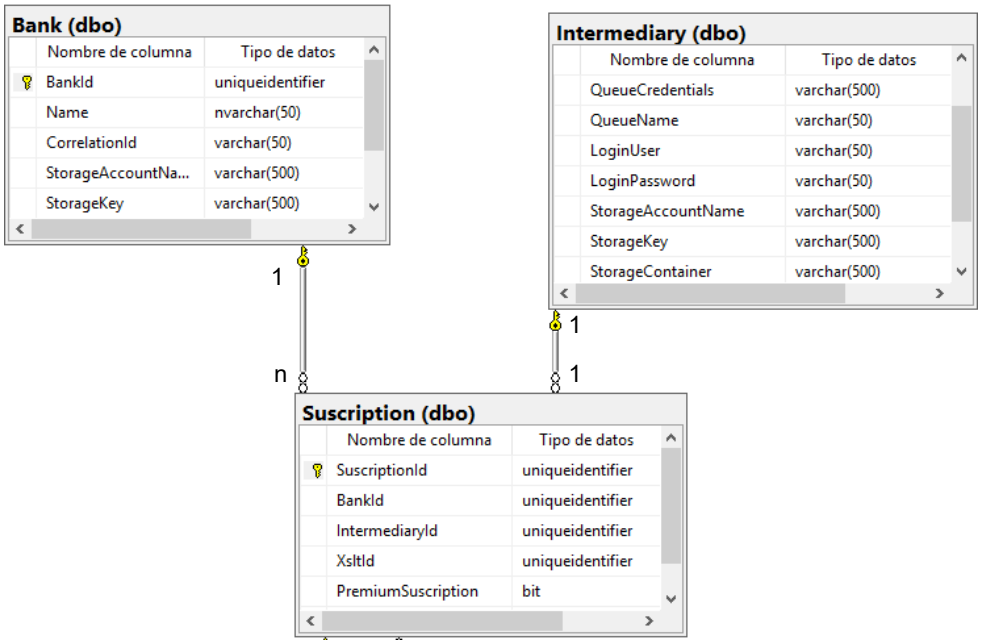


Figura 17: Tabla Relacional Suscription

Una vez se crea esta tabla intermedia, también estará apuntando a otra tabla llamada Xslt. Esta tabla básicamente almacena el código de las hojas de estilo XSLT para realizar la transformación de esa suscripción en concreto.

Una vez contextualizado el diseño de la base de datos, volvemos al flujo que explicábamos en la Fase 1.

Una vez la Function “Internal Message Processor” ha descargado el archivo con la información, es esta misma función quien inicia el proceso de transformación. No entraré en detalle con la explicación de esto, porque ya se explicará más ampliamente en el capítulo 6, pero el proceso consiste en rescatar de la tabla Xslt, la hoja de estilo necesaria para transformar el mensaje que estamos procesando.

Una vez se obtiene el contenido transformado se realizar una copia de seguridad en el sistema, registrando en las respectivas tablas tanto el mensaje transformado, como el mensaje sin transformar. Esto nos permitirá, por un lado, tener un control de todos los procesos que ha realizado el sistema y poder realizar correcciones en caso de error. Por otro lado, nos permite poder ofrecer al cliente la posibilidad de rescatar ese mensaje en caso de que el sistema del cliente lo pierda por alguna razón.

La Figura 18 nos muestra cuales son las relaciones de una Suscription, donde “una” Suscripción tiene “un” Xslt y “un” ProcessedMessage. “Un” UnprocessedMessage

tiene muchos ProcessedMessage. Con esto se viene a decir que, todo el proceso anteriormente mencionado se podrá repetir varias veces. Es decir, cada banco X que envía un mensaje, tendrá una lista de Intermediarios interesados en recibir ese mensaje convertido al formato que ellos desean. Por tanto, este proceso se repetirá hasta realizar la conversión del mensaje al formato de todos y cada uno de los suscritos a ese banco X.

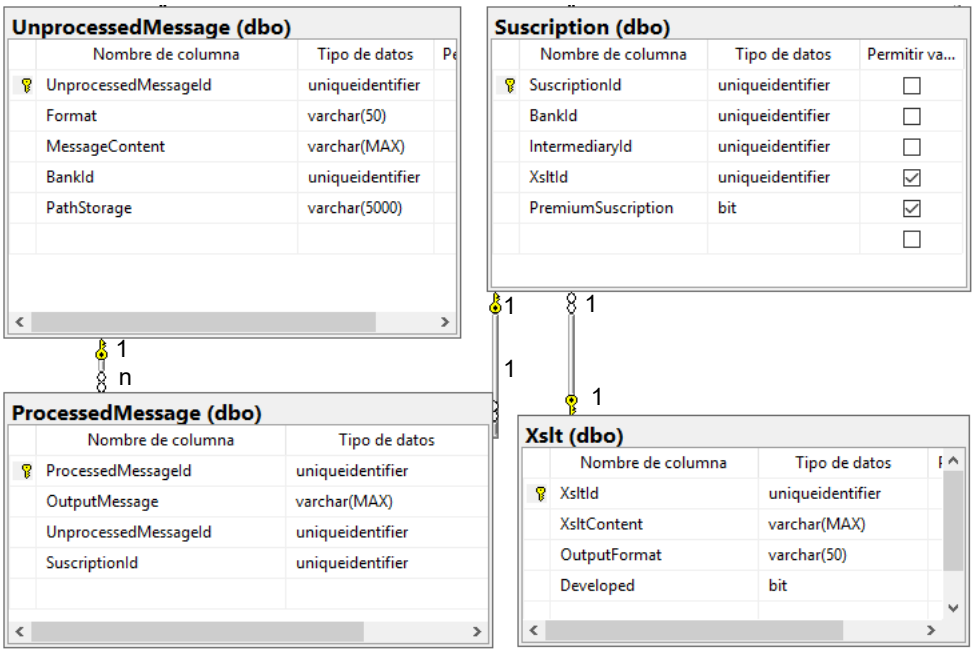


Figura 18: Relación de la Tablas

Como resumen de la fase 2, se podría decir que es una simple iteración entre la Function Processor y la base de datos, realizando hasta un total de 5 consultas e inserciones para finalmente conseguir realizar el proceso de transformación. Este proceso se repetirá para cada uno de los mensajes que se reciban en el sistema.

5.4 Fase 3: Almacenamiento y envío a intermediarios

El proceso realizado en la fase 3 es muy similar al realizado en la fase 1. Es decir, en esta fase enviaremos al intermediario el mensaje que él deseaba obtener, pero nos enfrentamos al mismo problema de la fase1; El contenido del mensaje es demasiado grande para poder enviarse mediante un servicio de mensajería de colas, así que

volvemos a utilizar un Blob Storage para subir el archivo y las colas para enviar la referencia de este archivo.

Cada uno de los intermediarios tendrá su propia cuenta de almacenamiento donde nuestro sistema subirá todos los archivos, clasificándolos automáticamente en carpetas dependiendo de la proveniencia del mensaje. Además, internamente estarán clasificados en base al formato del archivo como se puede apreciar en la Figura 19.

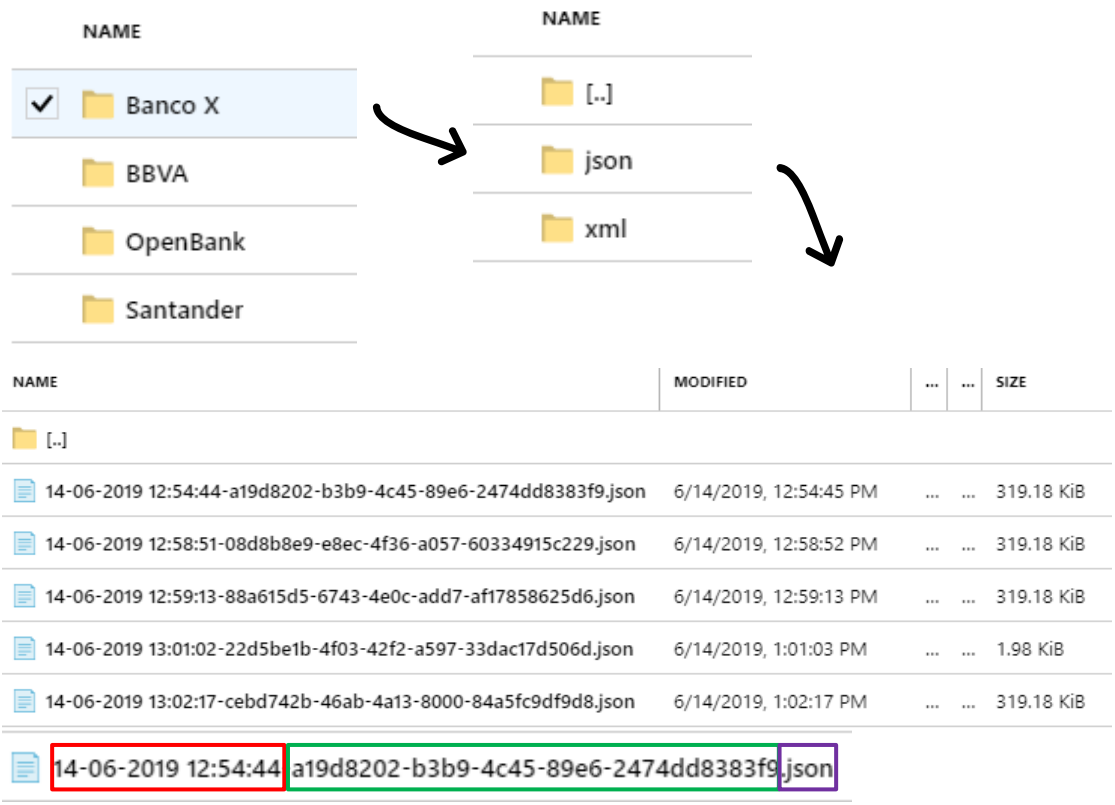


Figura 19: Sistema de clasificación de ficheros y formatos de archivos

El nombre del archivo se creará de la siguiente manera: primero la fecha de la transformación, a continuación, el identificador del mensaje ya transformado que se almacenó en nuestra base de datos y finalmente el formato del archivo (Figura 19). Crear el archivo con el identificador del mensaje procesado, nos permitirá localizar de manera rápida ese mensaje en caso de tener alguna incidencia con el cliente.

Una vez está subido el archivo en la nube, es necesario enviar al intermediario un mensaje con la ruta donde se encuentra el archivo final. Si que es verdad que el intermediario podría acceder al portal web de Azure, y mediante entorno gráfico navegar hasta llegar al archivo. Pero la intención es que automáticamente sus sistemas rescaten

el archivo, y para ello se envía un mensaje con la URL a ese archivo. Por tanto, el intermediario recibe un mensaje con el formato el siguiente formato (Figura 20)


```
"{
"bankName": "Santander",
"inputFormat": "xml",
"fileName": "14-06-2019 13:02:18-98d57726-140d-4e49-b6a2-3cdd3ddf3634.xml",
"URL": "https://finalstorage2.blob.core.windows.net/intermediarytestingstorage/Ban
}
```


Figura 20: Formato del mensaje que recibe el intermediario

El Blob Storage permite atender peticiones GET, como si de un servicio API REST se tratara, así que, con la URL, el sistema del intermediario podrá obtener el contenido del archivo.

Inicialmente nuestro sistema obtiene las credenciales, el nombre de la cuenta y toda la información necesaria para poder alojar el archivo convertido, en la cuenta del intermediario. Además, también se rescatará de la base de datos la información necesaria para poder encolar la URL del archivo en su cola respectiva.

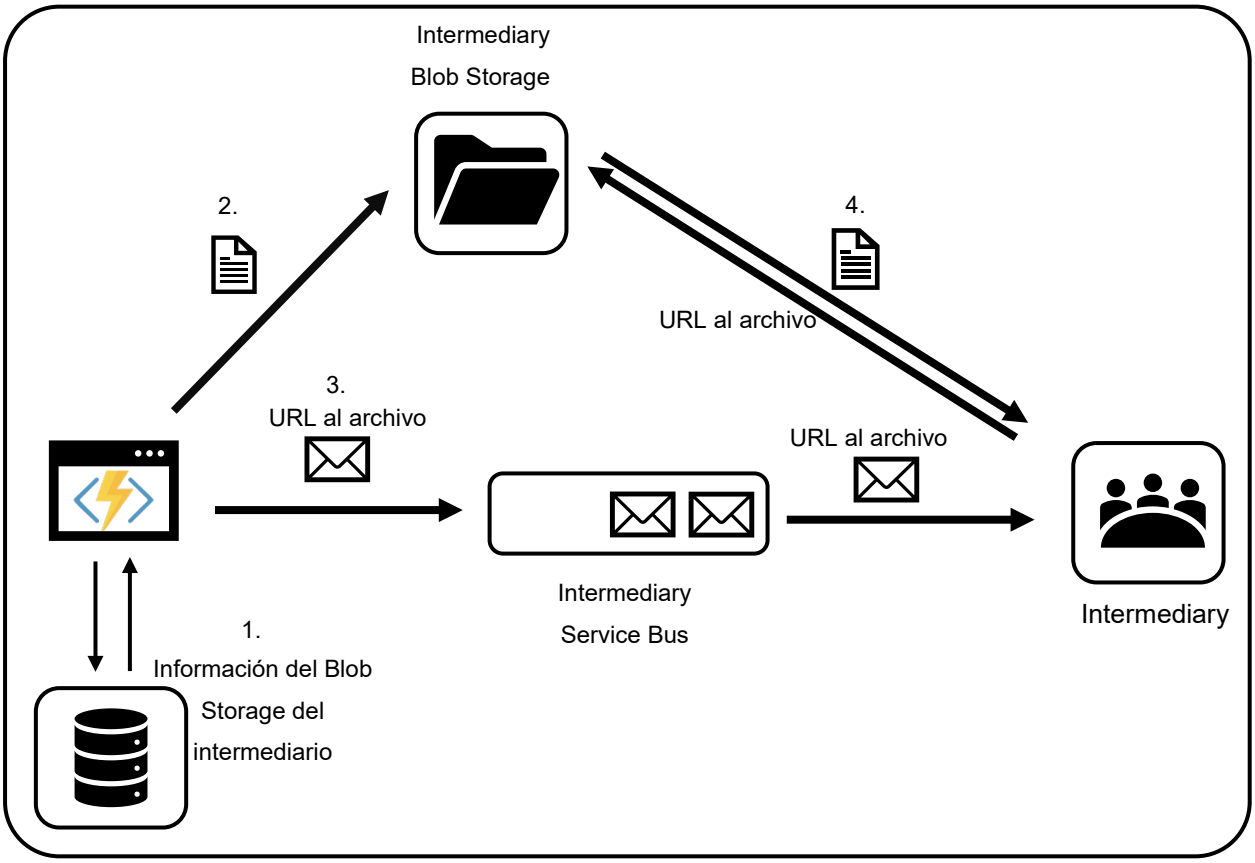


Figura 21: Proceso realizado en la Fase 3

El proceso de la Figura 21 se repetirá para cada uno de los Intermediarios que realizaron una suscripción sobre ese tipo de mensajes.

Finalmente, y a modo resumen, en la Figura 22, se pueden ver todos los componentes de nuestro sistema. Se puede pensar que, al ser la visión general del sistema, debería aparecer en el Capítulo 5.1 “Visión general del sistema”, pero intentar descifrar como funciona el sistema sin haber ido fragmentando el proceso paso a paso, puede resultar una dura tarea. Es por eso por lo que se ha puesto al final de todas las fases a modo resumen

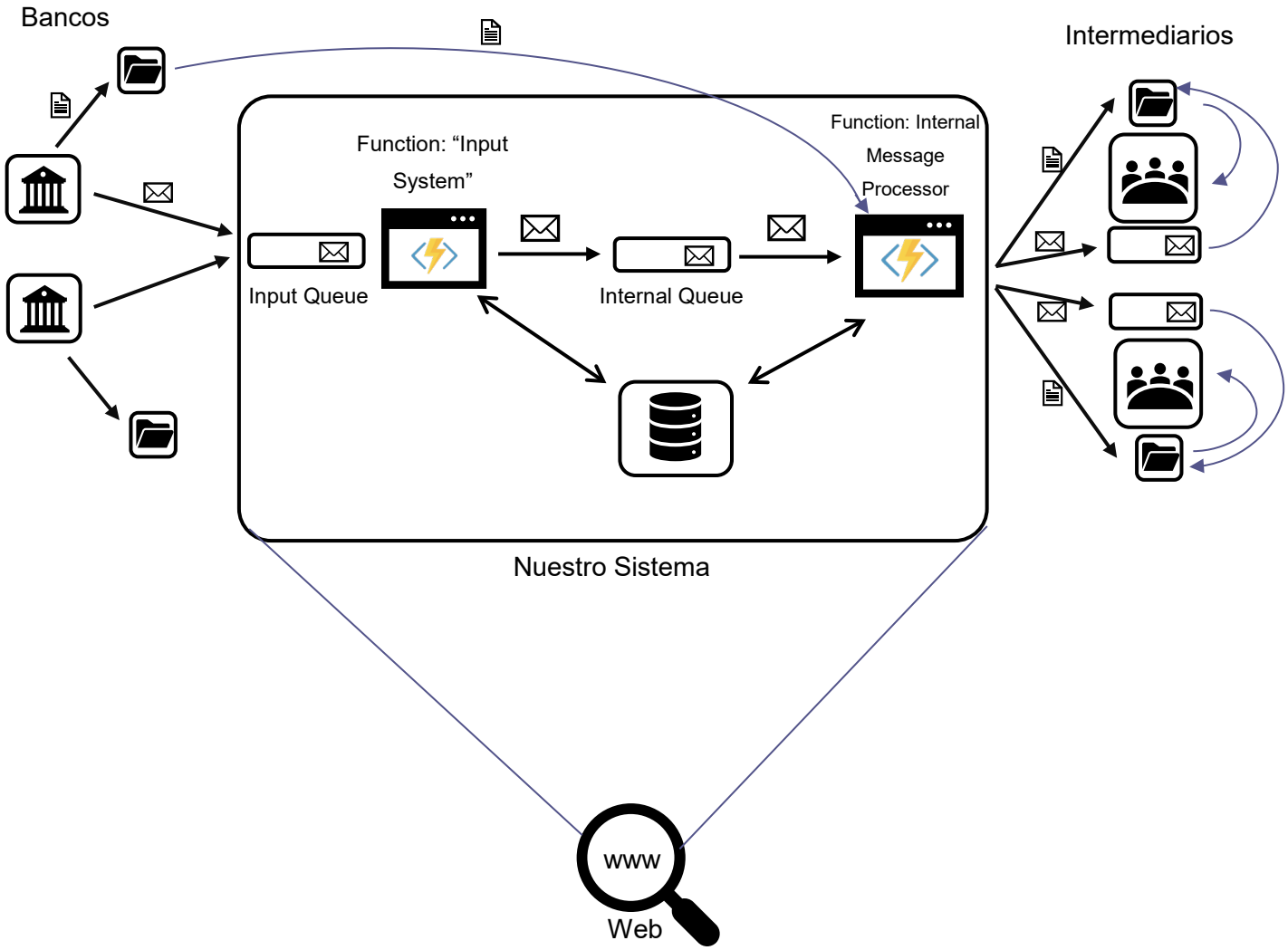


Figura 22: Visión General del Sistema

Capítulo 6

“Transformación”

6.1 Diseño sistema de conversión

Para hablar sobre el sistema de conversión le dedicaremos un capítulo entero; No por extensión, sino por complejidad temporal y porque realmente es la base de todo el proyecto; El conseguir transformar cientos de archivos, en otros completamente distintos de manera automática y rápida.

El sistema de conversión consta de 2 de componentes. Por un lado, la hoja de estilo XSLT, y por otro, el mensaje a transformar, obtenido del Blob Storage. Este mensaje, debido a un convenio, vendrá siempre en formato XML, ya que la gran mayoría de sistemas bancarios generan automáticamente estos archivos en este formato [17], y para nuestro sistema se simplifica la lógica al saber que siempre se obtiene así.

El sistema dispone de dos servicios de conversión. Por un lado, una transformación total del archivo de un formato a otro. Es decir, el archivo viene en formato XML y se desea convertir a formato JSON. Por otro lado, y es aquí donde radica una de las complejidades del trabajo, el sistema tiene la capacidad de realizar una transformación tanto de formato, como de lógica, es decir, cambiar valores e identificadores de forma personalizada, de manera que el intermediario obtenga un archivo a medida, pero en base a los valores originales.

Pero para identificar el tipo de conversión que se debe realizar, el sistema antes tiene que seguir un proceso.

Cada tabla “Suscription” tiene asociada una tabla “Xslt”. Cuando un banco envía un mensaje, se mirará la lista de cuantas Suscripciones hay a ese banco y para cada una de estas suscripciones se buscará la tabla XSLT asociada a esa suscripción, para saber que conversión realizar.

La clave para conocer si se trata de una conversión simple o personalizada está en el campo “PremiumSuscription” (Figura 23).

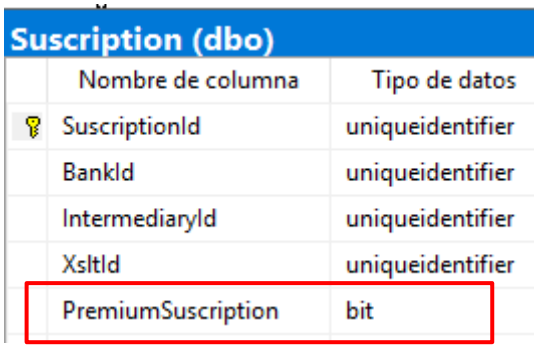


Figura 23: Tabla Suscription

Cuando el intermediario realiza una Suscripción a un banco, indicará el tipo de transformación que desea. Si indica una suscripción a medida, habría que ponerse en contacto con el intermediario para ver cuáles son sus necesidades y se crearía una hoja de estilo XSLT a medida.

6.2 Conversión estándar

En caso de que el campo PremiumSuscription sea Falso, el sistema simplemente consultará el formato de salida del archivo (Figura 24)

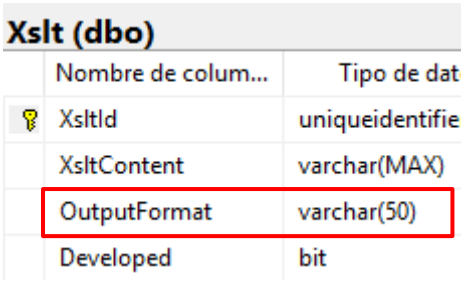


Figura 24: Tabla Xslt

En caso de que el formato final sea XML, se enviará al intermediario el mismo fichero sin realizar ninguna conversión, dado que el archivo ya venía en formato XML.

En caso de que el formato final sea JSON, utilizaremos diversas librerías como Xpath, XmlDocument, Linq, las cuales tienen diversas funcionalidades como, leer y mapear un archivo XML, dado que C# inicialmente interpreta el archivo como una simple

cadena de texto. Esta virtualización de archivo XML es lo que se llama árbol de nodos, como se ejemplifica en la Figura 25.

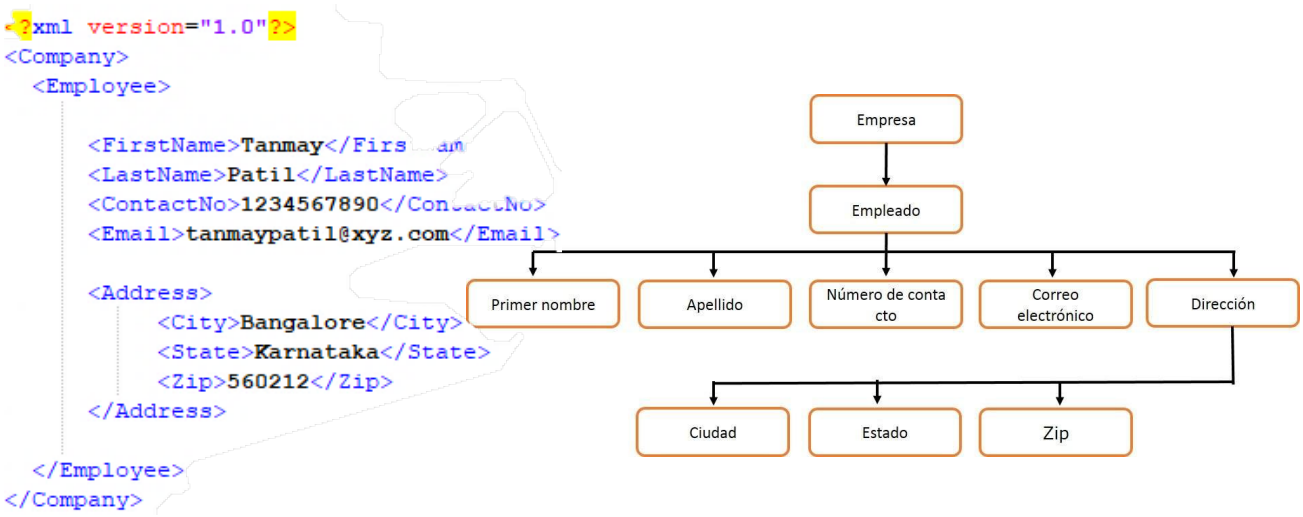


Figura 25: Árbol de nodos resultante del código XML

Una vez creada esa estructura simplemente habrá que usar la librería Newtonsoft, que, de manera automática, él convierte el árbol en formato JSON. Un ejemplo de conversión de formato XML a JSON es el siguiente (Figura 26).

Figura 26:Conversión de formato XML a JSON

6.3 Conversión Personalizada

Por otro lado, cuando el cliente requiere de una conversión más avanzada donde no solo se modifica el formato, se requerirá de la creación de hojas de estilo XSLT. Inicialmente el usuario solicitará la creación de esta hoja, pero para evitar fallos, por

cuestiones técnicas estará temporalmente deshabilitada esa conversión personalizada hasta que se desarrolle por completo, y mientras tanto será una conversión estándar. Esto lo conseguimos gracias al campo Developed (Figura 27).

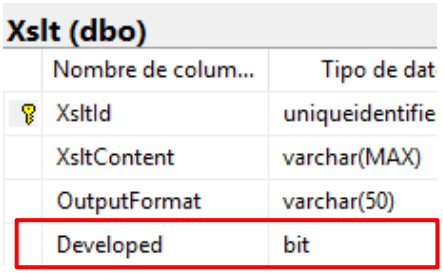


Figura 27: Tabla Xslt

Una vez esté desarrollado el XSLT, el administrador lo insertará en Base de datos y activará el bit de desarrollado, comenzando ya a funcionar.

Los archivos XSL son un conjunto de reglas, que permiten recorrer todos los nodos de un archivo XML, comprobando el nombre del nodo y su valor. De esta manera podemos ir "pintando" un nuevo documento en base a las reglas [13].

XSLT es una combinación de los siguientes elementos:

- XSLT: es el lenguaje para transformar documentos XML
- XPath: es el lenguaje para navegar por documentos XML
- XQuery: es el lenguaje para generar reglas en base a los valores del documento XML

Este lenguaje nos permite realizar operaciones lógicas básicas como "Value-Of", "For-Each", "if", "choose" y "sort". Más allá de esas operaciones lógicas, el resto de funcionalidades son para tratar con cadenas de texto, con fechas o con números.

Cabe mencionar una de las grandes problemáticas de este lenguaje, y por el cual se hace tan complicado realizar lógicas complejas; Para realizar lógicas sencillas es muy sencillo de usar, pero a la hora de realizar lógicas complejas permite almacenar variables, pero no permite cambiar dinámicamente su valor. Esto genera, que el simple hecho de crear un contador para realizar un bucle, sea imposible, teniendo que buscar complejas alternativas, llegando a casos en los que es imposible realizar determinadas lógicas, usando las técnicas tradicionales.

Tras mencionar la dificultad que supone usar hojas de estilo XSLT, se puede hacer la pregunta de porque usar este método. Pues la respuesta es sencilla, su mayor defecto, la simpleza, se convierte en su mayor ventaja; Es decir, debido a que son reglas muy básicas y ejecutadas de manera lineal, la velocidad para convertir un fichero en otro es muy elevada. Y esto último es un factor muy a tener en cuenta, pues se reciben archivos con gran cantidad de información, y de no ser así el sistema necesitaría mucha mayor potencia computacional. Por ejemplo, el sistema llega a procesar en ocasiones ficheros con casi 7.000 líneas, y una gran cantidad de nodos anidados (Figura 28). Esto último supone una dificultad extra también a la hora de programar la lógica, pues puede suponer un problema el tener recorrer los nodos entrando y saliendo a un nodo más interno de manera dinámica.

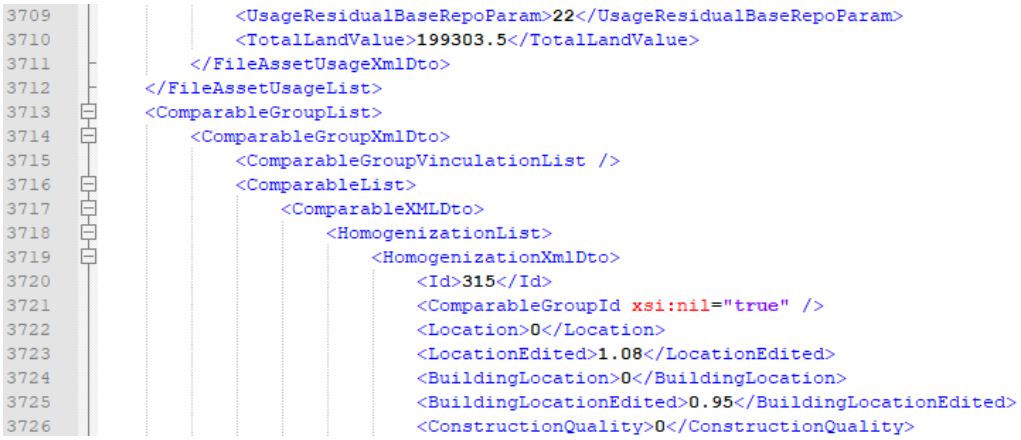


Figura 28: Fichero XML con gran cantidad de nodos y líneas de código

Una vez expuesto todo esto, vamos a ejemplificar para demostrarlo, y, además ver la mecánica de trabajo a la hora de realizar las hojas de estilo. Empezamos con un caso muy básico, donde explicaremos cada uno de los elementos importantes (Figura 29).

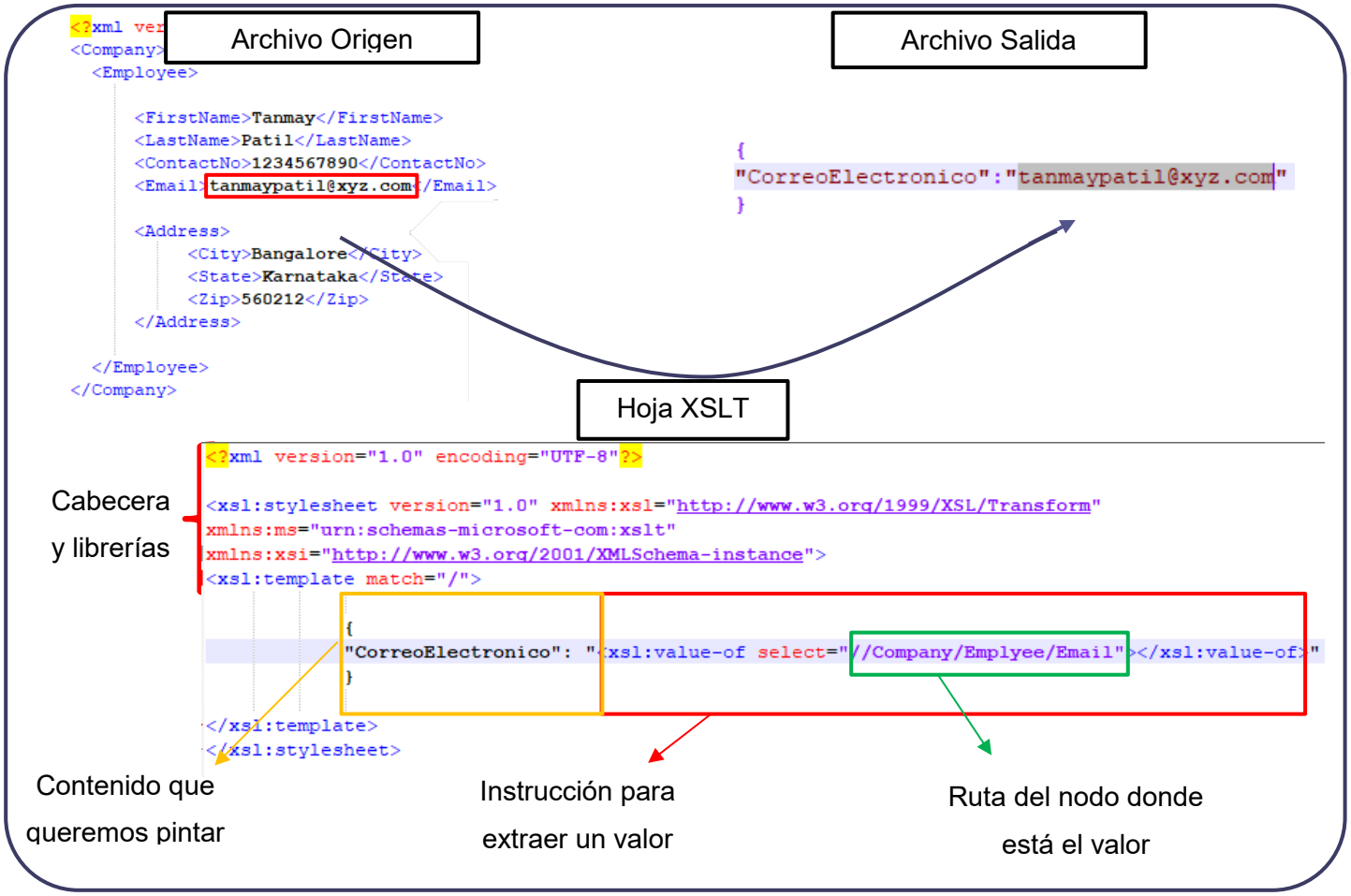


Figura 29: Ejemplo de conversión con XSLT

Con la figura anterior podemos hacernos una idea del fin que se persigue y como conseguirlo. Así que solo habría que seguir las especificaciones que tiene el cliente. El modo de trabajo seguido es crear un boletín (Figura 30), indicando los nodos de extracción, las condiciones lógicas y el resultado final, para posteriormente desarrollar el archivo.

	Localización Nodo XML	JSON (Destino)		Tipo de Transformación		Valor Final
3			"assessmentData":{			
4	<FileVal><ProductType>		verification	Regla Simple		Si el valor del nodo es 01, enviamos al cliente 0. Si el valor del nodo es 05, enviamos al cliente 7.
5			valuerAmount	Fijo		0.00
6	<PredefiniedFields><FechaSupervision>		emissionDate	Directo		dd/mm/yyyy
7	<FileClassGroupList><FileClassGroupXmlDto><ITEReportParam>		tecnicInspection	Regla Simple	For each <FileClassGroupXmlDto><ITEReportParam>	Si no existe nodo O todos los nodos tienen valor 3 devolvemos null.Si todos los nodos tienen valor 1 se devuelve true, en caso contrario se devuelve false.
8	<FileClassGroupList><FileClassGroupXmlDto><StructureParam>		structure	Regla Compuesta	Si no existe el nodo enviar null	Ver Anexo Structure
9	<FileClassGroupList><FileClassGroupXmlDto><ShortComingsParam>		deficCorrected	Bucle	Este nodo es obligatorio sólo cuando el valor tecnicInspection tiene el valor false. Si es true, este valor NO se notifica.	Si todos los nodos tienen valor 1 se devuelve true, en caso contrario se devuelve false.

Figura 30: Guía general de criterios y condiciones

Como se puede observar en la tabla hay varios tipos de transformación; Desde valores directos y fijos, hasta reglas un poco más complejas que necesitan un anexo especial, como la Fila 8 de la Tabla, para explicar correctamente el valor a recibir y como conseguirlo. Hay casos sencillos, donde el desarrollo del código se podrá transformar de pseudo código a código XSLT. A continuación, un ejemplo real conversión con su respectivo código, en la Figura 31 se ven los requerimientos y en la Figura 32 el código XSLT que conseguirá obtener el valor deseado.

1. Para cada nodo <FileClassGroupXmlDto> será necesario realizar la siguiente correspondencia:

iqueta StructureParam es un 7 será necesario comprobar el valor de la etiqueta I
- Si existe al menos un elemento con madera se enviará
- En caso de existir, prefabricada, como 4 Prefabricada
- Si existe hormigón, muro o metálica (modificado 25/09/2017) como 1 hormigón
- Y resto como 5 otros

Origen						Destino	
Etiqueta Origen	Valor	Descripción	Etiqueta 2 XML	Valor (*)	Descripción	Valor	Descripción
StructureParam	1	Por muros de carga				1	Hormigón
	2	Por pórticos de hormigón armado				1	Hormigón
	3	Por pórticos de acero laminado				1	Hormigón
	4	Por entramado de madera				2	Madera
	5	Metálica y de hormigón				1	Hormigón
	6	Prefabricada				4	Prefabricada
			Mixed Structure Param	1	Muros	1	Hormigón
				2	Hormigón	1	Hormigón
				3	Metálica	1	Hormigón
	7	Mixta =>		4	Prefabricad	4	Prefabricada
				5	Otros	5	Otros
				6		3	Mixta Mader

Figura 31: Requerimientos para el campo "Structure"

```
<xsl:choose>
  <xsl:when test="(//FileClassGroupList/FileClassGroupXmlDto/StructureParam = '1'
                                        (//FileClassGroupList/FileClassGroupXmlDto/St
    "structure":1,
  </xsl:when>
  <xsl:when test="//FileClassGroupList/FileClassGroupXmlDto/StructureParam = '4''
    "structure":2,
  </xsl:when>
  <xsl:when test="//FileClassGroupList/FileClassGroupXmlDto/StructureParam = '6''
    "structure":4,
  </xsl:when>
  <xsl:when test="//FileClassGroupList/FileClassGroupXmlDto/StructureParam = '7''
    <xsl:for-each select="//FileClassGroupList/FileClassGroupXmlDto/MixedStructur
      <xsl:variable name="mixStructPar" select="//FileClassGroupList/FileClassGro
      <xsl:if test="not(. = $mixStructPar)">
        <xsl:choose>
          <xsl:when test="$mixStructPar = 4">
    "structure":4,
          </xsl:when>
          <xsl:when test="$mixStructPar = 3">
    "structure":3,
          </xsl:when>
          <xsl:when test="$mixStructPar = 2">
    "structure":2,
          </xsl:when>
          <xsl:when test="$mixStructPar = 1">
    "structure":1,
          </xsl:when>
          <xsl:when test="$mixStructPar = 5">
    "structure":5,
          </xsl:when>
          <xsl:otherwise>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:if>
    </xsl:for-each>
    <xsl:if test="(//MixedStructureParam = '01') or (//MixedStructureParam = '02'
    "structure":1,
    </xsl:if>
    <xsl:if test="//MixedStructureParam = '04'">
    "structure":4,
    </xsl:if>
    <xsl:if test="//MixedStructureParam = '05'">
    "structure":5,
    </xsl:if>
    <xsl:if test="//MixedStructureParam = '06'">
    "structure":3,
    </xsl:if>
  </xsl:when>
  <xsl:otherwise>
    "structure":null,
  </xsl:otherwise>
</xsl:choose>
```

Figura 32: Código XSLT para transformar el campo Structure

Como se puede observar, se pueden llegar a realizar grandes fragmentos de código para simplemente extraer un nuevo campo <clave>:<valor>. Esto supone un alto coste temporal para desarrollar una hoja de estilo con muchos campos. Por eso el desarrollo de conversores es una de las partes más costosas tanto temporalmente como técnicamente, pues, aunque el ejemplo de la figura anterior son simples comparaciones y es sencillo de realizar, hay otros casos como la realización de bucles o consulta de nodos cuya dirección es dinámica, supone un gran reto.

Ejemplificar supondría un exceso de imágenes, así que el archivo a convertir [18], los código de los conversores [19] [20] y el boletín de conversión [21] lo pongo en el anexo.

Capítulo 7

“Aplicación Web”

7.1 Uso y funcionalidad

Esta parte del proyecto se centra en conseguir mostrar de manera visual todos los entresijos de la integración, y dotarlo de funcionalidad, haciéndolo todas las gestiones del sistema mucho más sencillo. Para explicar la finalidad de la web, lo podemos dividir en dos enfoques. El usuario (intermediario) y el administrador.

Por un lado, el intermediario, tendrá la opción de registrarse en el sistema, rellenando todos los campos necesarios para el correcto funcionamiento de las transacciones y transformaciones que se realizan por detrás.

Una vez se haya registrado, nuestros técnicos verificarán a este usuario, por cuestiones técnicas y legales, y habilitarán el acceso de este usuario. Desde dentro del menú del usuario corriente, se permitirá modificar toda su información personal (Figura 33).

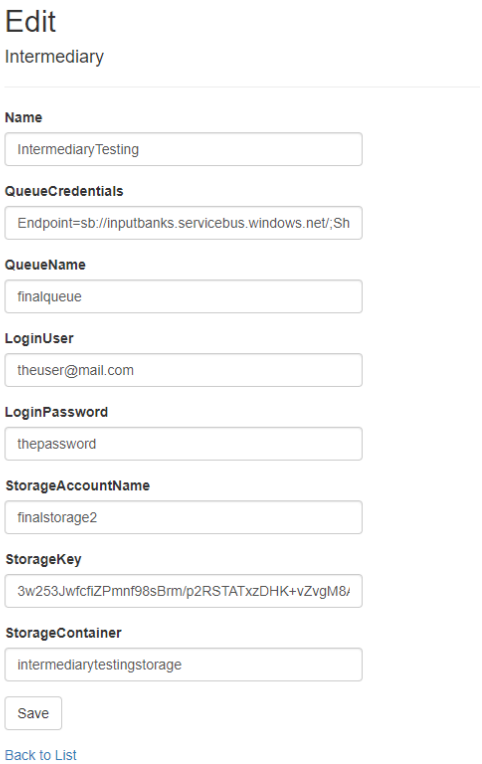


Figura 33: Menú para editar información personal

Sobre todo, y la función principal de esta web es la de poder permitir al usuario elegir de quien obtener los datos, o si desea recibir una conversión personalizada, es por ello que existe un menú donde poder realizar una suscripción a banco deseado, como se puede observar en la Figura 34.

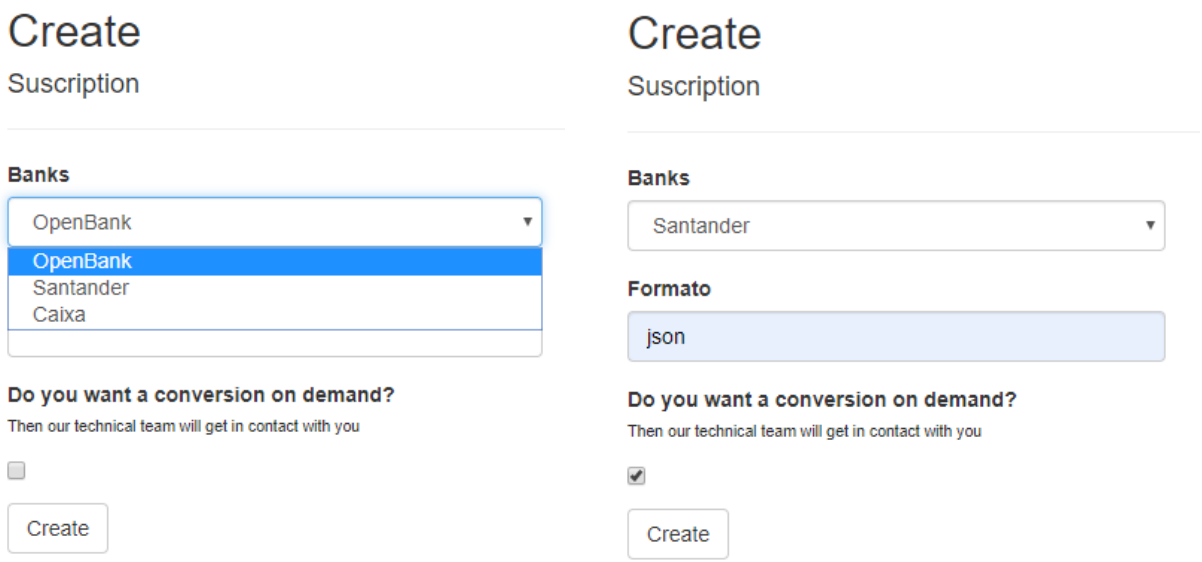


Figura 34: Menú de creación de nueva Suscripción a un Banco

Además, el usuario tendrá una lista con todas las "suscripciones" de los bancos a de los que está recibiendo mensajes, y se le permitirá realizar las operaciones básicas con cada una de ellas, como eliminar, modificar o ver todos los detalles. En la Figura 35, se puede observar como en la lista con las suscripciones de un cliente, aparece un banco repetido. Esto es porque puede suceder que un intermediario desee obtener la misma información en distintos formatos.

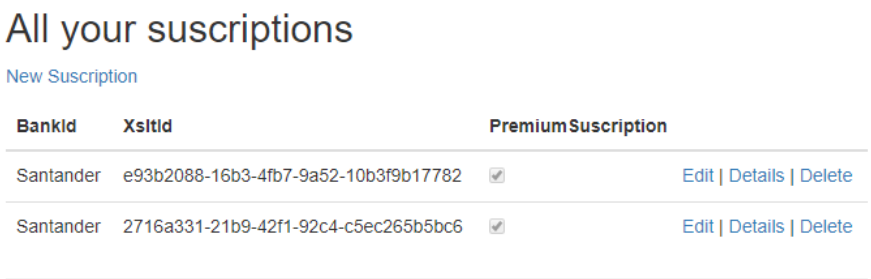


Figura 35: Lista de Suscripciones de un Intermediario

Por otro lado, el administrador accederá a su usuario, y tendrá acceso a realizar todas las gestiones necesarias como ver y modificar todos los usuarios, bancos, suscripciones, conversores…(Figura 36)

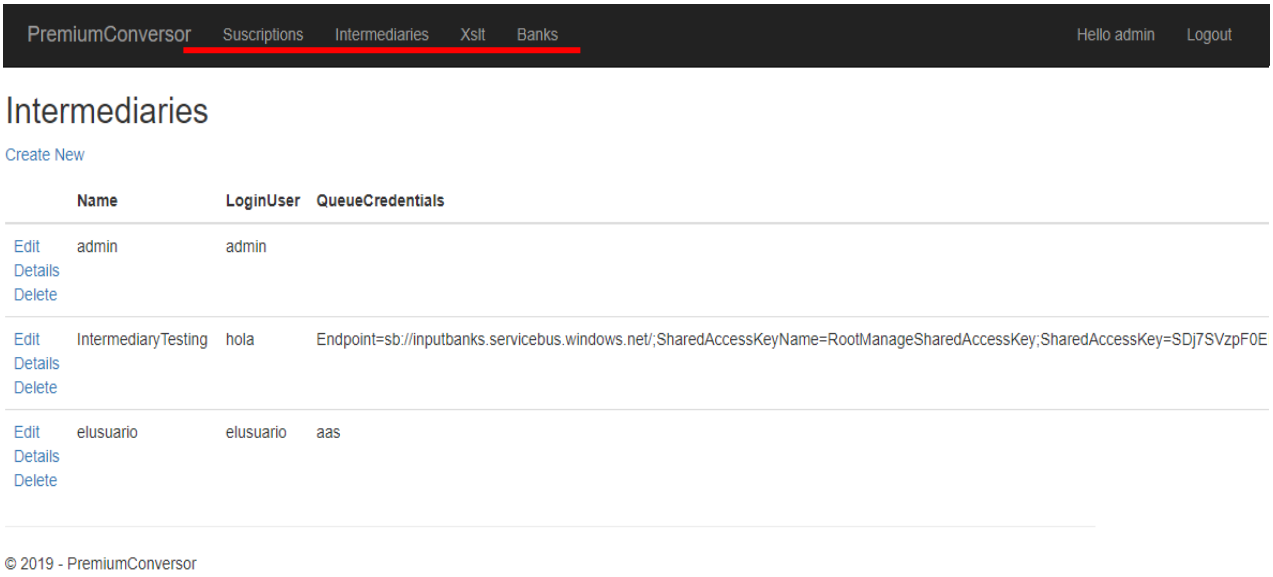


Figura 36: Menú del Administrador

Además es importante mencionar una característica muy importante. Es **Responsive,** es decir, es capaz de adaptarse a los distintos dispositivos donde se vaya usar, como PCs, tablets, IOs y dispositivos android. En la Figura 37 se puede contemplar como se adapta a Tablet y dispositivo movil.

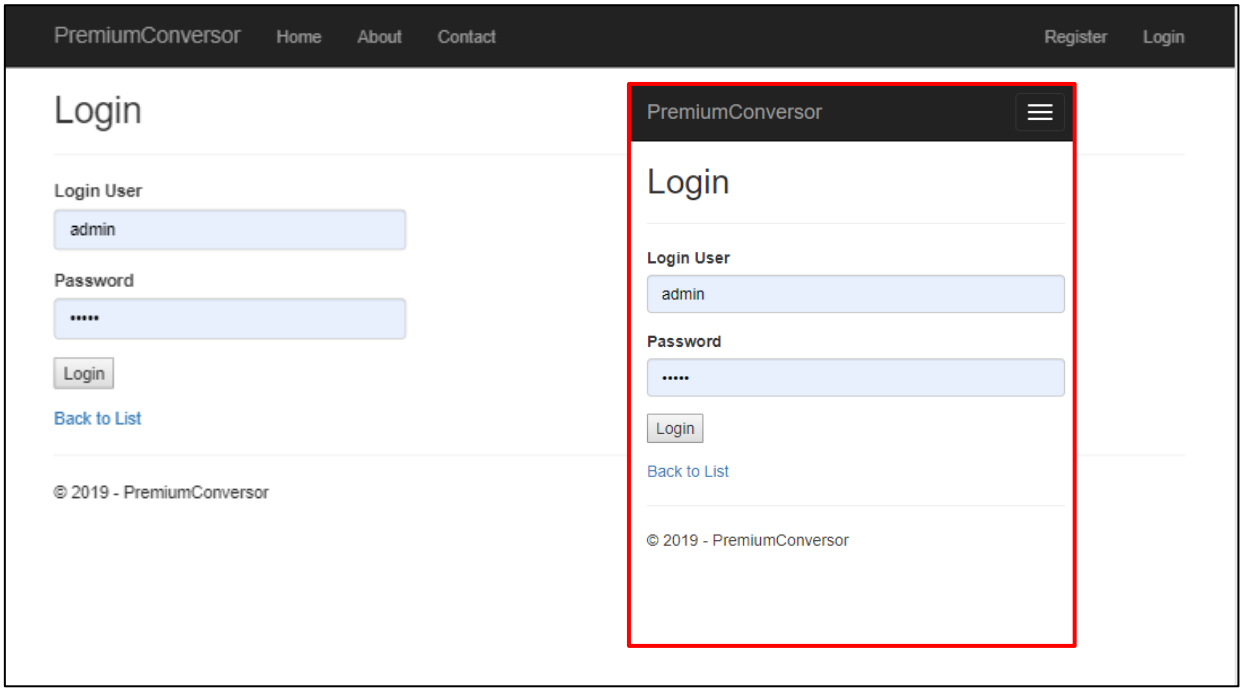
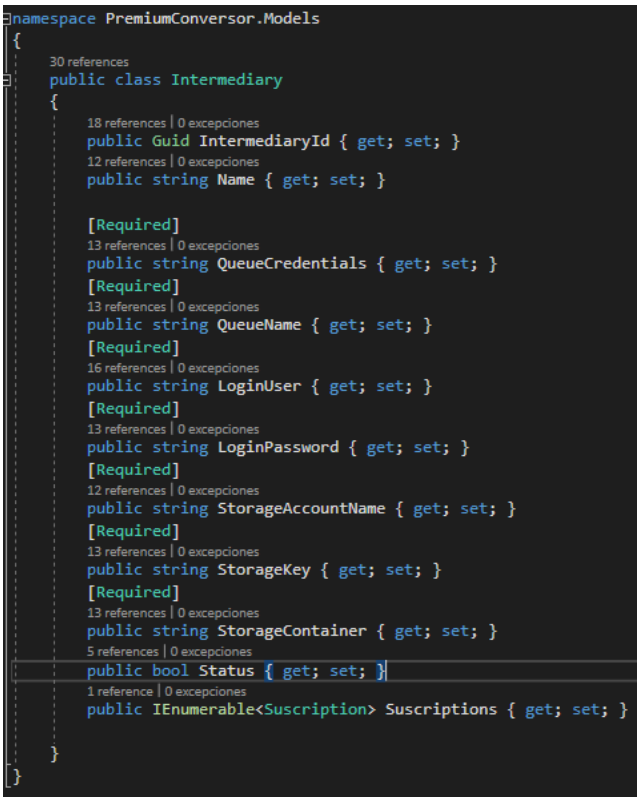


Figura 37: Diseño web en versión table y móvil

7.2 Implementación

El desarrollo integro de esta web ha sido mediante ASP .NET CORE. Esta aplicación web está basada en el modelo MVC (Modelo, Vista, Controlador). Para ello se realizaron todos los modelos de cada una de las tablas de nuestra base de datos. En la Figura 37, se ver como para la tabla Intermediary, se crearon todas las columnas, además de las relaciones con otras tablas



```
namespace PremiumConversor.Models
{
    30 references
    public class Intermediary
    {
        18 references | 0 excepciones
        public Guid IntermediaryId { get; set; }
        12 references | 0 excepciones
        public string Name { get; set; }

        [Required]
        13 references | 0 excepciones
        public string QueueCredentials { get; set; }
        [Required]
        13 references | 0 excepciones
        public string QueueName { get; set; }
        [Required]
        16 references | 0 excepciones
        public string LoginUser { get; set; }
        [Required]
        13 references | 0 excepciones
        public string LoginPassword { get; set; }
        [Required]
        12 references | 0 excepciones
        public string StorageAccountName { get; set; }
        [Required]
        13 references | 0 excepciones
        public string StorageKey { get; set; }
        [Required]
        13 references | 0 excepciones
        public string StorageContainer { get; set; }
        5 references | 0 excepciones
        public bool Status { get; set; }
        1 reference | 0 excepciones
        public IEnumerable<Suscription> Suscriptions { get; set; }

    }
}
```

Figura 38: Modelo de la tabla Intermediary

Una vez creados los modelos, las relaciones entre ellos, y los repositorios con todos los métodos a invocar, creamos las vistas y los controladores básicos gracias a la técnica de Sacaffolding, que se encarga de generar todas las vistas y controladores de cada uno de los modelos. De este modo tenemos un menú de crear, eliminar, editar, y listar de cada una de las tablas de la base de datos.

Utilizando esto como base, se ha ido modificando para que cumpliese los requerimientos establecidos.

Por otro lado, una de las mayores complicaciones que se han encontrado para este proyecto fue la de la creación de un Login, pues por defecto, Visual Studio crea un modelo de Login bastante avanzado con grandes medidas de seguridad, pero este realizaba todas las operaciones con una base de datos propia. Para nuestro caso necesitábamos registrar toda la información en nuestra base de datos para posteriormente realizar las operaciones pertinentes a la hora de enviar mensajes, y realizar las transformaciones. Así que finalmente se optó por la creación de un Login, basado en Sesiones, guardando en la memoria virtual del navegador la información del usuario que se encuentra navegando. En caso de inactividad del usuario, cada X cantidad de tiempo, se eliminará de memoria, y se expulsará al usuario, necesitando loguearse otra vez. En la Figura 39, se puede observar la línea de código que estable que las sesiones serán de 10 minutos (de inactividad).

```
services.AddSession(options => {
    options.IdleTimeout = TimeSpan.FromMinutes(10);//You can set Time
});
```

Figura 39: Lapso de tiempo que dura una Sesión en nuestro sistema

Por último, mencionar como se ha realizado el alojamiento de todo este código. Azure permite de manera muy sencilla publicar este servicio, alojarlo y tener un control de rendimiento de esta web. Todo esto se realiza creando un App Service y un Plan de alojamiento que apenas necesita configuración. El coste dependerá del rendimiento que necesitemos y del tiempo que esté en funcionamiento, pues podemos encender y apagar los servicios como si de un dispositivo móvil se tratase. La URL para poder acceder a esta web es: *https://premiumconversor.azurewebsites.net*

Capítulo 8

“Conclusiones”

Para concluir este trabajo final de grado, cabe destacar que se han cumplido con éxito todas las tareas, y sobre todo el objetivo final: desarrollar una solución de integración entre sistemas bancarios e intermediarios.

Como se ha comprobado en el capítulo 6 se ha conseguido implementar y diseñar una base de datos que se adapta a las necesidades de nuestro sistema. Además de crear todos los subprocesos necesarios para conseguir encaminar todos los mensajes a los sitios deseados.

También se ha conseguido desarrollar un sistema capaz de, por un lado, convertir por completo un archivo XML a JSON, y por otro lado realizar las transformaciones personalizadas, gracias a que se han desarrollado 2 hojas XSLT una de XML a JSON y otra de XML a XML. Además, se ha creado un sistema capaz de reconocer de forma automática que tipo de conversión debe realizar

Finalmente, y aunque con alguna dificultad, se ha completado la web. La dificultad proviene de una planificación inicial en la que la web se iba a diseñar a partir de una plantilla HTML5 y PHP. Finalmente, por funcionalidad e integración con el resto de las herramientas del sistema de conversión, se decidió implementar la web con ASP .Net Core MVC.

De cara a una implementación futura, habría que estudiar bien el campo legislativo, debido a que la distribución de información bancaria relacionada con sus clientes, tiene fuertes controles debido a la Ley De Protección de Datos: **Reglamento de la Unión Europea 2016/679 de 27 de abril de 2016 relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos, así como en la Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales.**

Por esta razón, tanto bancos, como intermediarios deberían modificar su política de condiciones, siendo los clientes quienes tienen el control total de sus datos y autorizan este tipo de distribución (Reglamento de la Unión Europea 2016/679 de 27 de

abril de 2016, *Artículos 5, 6, 7,13,17,18,24*) [22], pudiendo incentivar al cliente con ciertas ventajas al autorizar el funcionamiento de este sistema. Por nuestra parte debido a que nuestra actividad simplemente se basa en procesarlo y transportarla, deberíamos cumplir las medidas de seguridad que cada uno de los bancos nos solicitasen (Reglamento de la Unión Europea 2016/679 de 27 de abril de 2016, *Artículos 30, 35, 45)* [22].

Finalmente, y como opinión personal, creo que no solo se han alcanzado los objetivos iniciales, sino que, debido a la metodología incremental, se han conseguido realizar muchas pequeñas implementaciones y mejoras en el sistema, que sobre la planificación inicial no se planteaban. Además de haber realizado casi íntegramente todo el trabajo con herramientas nunca vistas en la universidad (aunque con similar base teórica) permitiéndome un gran aprendizaje de manera autodidacta de unas herramientas que, todo apunta, serán las herramientas usada en el mañana.

Referencias

[1] Y. A. Abreu, «Tipos de Fragmentación de Bases de Datos Distribuidas» 01 2006. Consultado en: https://www.redalyc.org/html/3604/360433560015/. [Último acceso: 23 06 2019].

[2] TECON, «¿Que es Microsoft Azure?,» Consultado en: https://www.tecon.es/que-es-microsoft-azure-como-funciona/. [Último acceso: 23 06 2019].

[3] P. V. A. J. Fons i Cors, «Tema 1 Asignatura IAP,» de *Tema 1 Asignatura IAP*, 2019.

[4] P. V. A. Joan Fons i Cors, «Tema 3 Asignatura IAP,» 2019.

[5] «Serverless Stack». Consultado en: https://serverless-stack.com/chapters/es/what-is-serverless.html. [Último acceso: 06 2019].

[6] «Enterprise service bus» 30 04 2019. Consultado en: https://es.wikipedia.org/wiki/Enterprise_service_bus#cite_ref-2. [Último acceso: 23 06 2019].

[7] D. Chappell, Enterprise Service Bus, O'Reilly: ISBN 0-596-00675-6, Junio 2004.

[8] «Microsoft Azure Service Bus» Consultado en: https://azure.microsoft.com/es-es/pricing/details/service-bus/. [Último acceso: 23 06 2019].

[9] «Microsoft Azure SQL Database» 10 05 2019. Consultado en: https://docs.microsoft.com/es-es/azure/sql-database/sql-database-features. [Último acceso: 23 06 2019].

[10] A. Silberschatz, Fundamentos de bases de datos, McGRAW-HILL.

[11] «Extensible Markup Language». Consultado en: https://es.wikipedia.org/wiki/Extensible_Markup_Language#cite_note-1. [Último acceso: 23 06 2019].

[12] «JSON» 09 05 2019. Consultado en: https://es.wikipedia.org/wiki/JSON. [Último acceso: 23 06 2019].

[13] «W3C» 23 01 2007. Consultado en: https://www.w3.org/TR/xslt20/#xslt-mime-definition. [Último acceso: 23 06 2019].

[14] «OBS Business School». Consultado en: https://www.obs-edu.com/es/blog-project-management/metodologias-agiles/caracteristicas-y-fases-del-modelo-incremental.

[15] «Modelo incremental,» 29 03 2016. Consultado en: http://marich.blogspot.es/1459223366/modelo-incremental/. [Último acceso: 23 06 2019].

[16] «OBS Business School,» Consultado en: https://www.obs-edu.com/es/blog-project-management/temas-actuales-de-project-management/metodo-kaizen-aplicacion-y-beneficios. [Último acceso: 23 06 2019].

[17] SEPA, «CECA,» 20 Noviembre 2015. Consultado en: https://www.ceca.es/wp-content/themes/ceca/assets/Cuaderno%20XML%20SCT%20Noviembre%202015.pdf. [Último acceso: 08 Junio 2019].

[18] C. Negrillo, «Mensaje Modelo XML,» Consultado en: https://bankstorage.blob.core.windows.net/xmlmessages/ModificadoSin.xml.

[19] C. Negrillo, «XSLT to JSON,» 20 04 2019. Consultado en: https://bankstorage.blob.core.windows.net/xmlmessages/converJSON.xslt.

[20] C. Negrillo, «XSLT to XML,» 25 04 2019. Consultado en: https://bankstorage.blob.core.windows.net/xmlmessages/converXML.xslt.

[21] C. Negrill, «Boletin de conversión,» [En línea]. Available: https://bankstorage.blob.core.windows.net/xmlmessages/Mapeo-Boletín.xlsx.

[22] «Reglamento de la Union Europea 2016/679,» 27 04 2016. [En línea]. Available: https://eur-lex.europa.eu/legal-content/ES/TXT/PDF/?uri=CELEX:02016R0679-20160504&from=EN.