**Universidad Complutense de Madrid**

**Facultad de Informática**


# Inclusión de un repositorio de datos fijos en un sistema de tesorería bancaria

Alumno: **Óscar López-Laguna Ortiz**

Directores de proyecto: **José Luis Risco Martín**

**Josué Pagán Ortiz**

Trabajo de Fin de Grado

Grado en Ingeniería del Software

Septiembre de 2017

# Índice general

# Palabras clave

## Palabras clave en Español


- Tesorería bancaria
- Datos fijos
- SOA
- SOAP
- Servicio Web
- WSO2


## Keywords in English


- Bank's treasury
- Fixed data
- SOA
- SOAP
- Web service
- WSO2

# Resumen

## Resumen en Español

La Tesorería de una entidad bancaria se enmarca dentro del negocio de banca de inversión (también denominada banca mayorista o corporativa) así como otros negocios tales como el de fusiones y adquisiciones, financiación de proyectos, préstamos sindicados o banca transaccional global. Pero, sin duda alguna, la Tesorería, o simplemente área de mercados, es una pieza fundamental dentro de la estructura de la banca de inversión de un banco.

La Tesorería de una Entidad está en constante comunicación con todos los departamentos, tanto internos como externos. Es por ello que todo el mundo financiero se mueve en torno a las acciones y decisiones que se llevan a cabo en la Tesorería. El área de la tesorería bancaria maneja, a su vez, grandes cantidades de datos para poder gestionar las operaciones, la contratación de productos y las confirmaciones y liquidaciones.

Actualmente, en la mayoría de entidades financieras, los datos están siendo gestionados por distintos aplicativos. Cada uno de estos aplicativos está desarrollado en distintos lenguajes de programación como COBOL, C++, Java... Esto supone costes y pérdidas de tiempo para las entidades a la hora de implementar intercambios de información entre las aplicaciones.

El trabajo de fin de grado se centra en la definición de una arquitectura de aplicativo capaz de gestionar los datos fijos utilizados dentro de la tesorería bancaria e integrarlo dentro de una arquitectura orientada a servicios (SOA). Estos datos están en constante crecimiento en función de las necesidades de los consumidores.

Se implementará un aplicativo de ejemplo para validar la arquitectura propuesta, que sea capaz de manejar las entidades de datos más importantes a gestionar dentro de la tesorería, como son:

- Contrapartidas, que comprenden:
  - Atributos generales
  - Contratos
  - Diccionarios
  - Contactos
  - Confirmaciones
  - Datos regulatorios
  - Instrucciones de liquidación
- Estructura interna, que comprende:
  - Portfolios
  - Grupos
  - Oficinas
- Convenios de mercado, que comprende:
  - Índices
  - Divisas
  - Calendarios
  - Unidades Geográficas

## Summary in English

The Bank's treasury is framed under the investment banking business, aka wholesale banking or global corporate banking, so do other areas as mergers and acquisitions, project funding, syndicated loans or global transactional banking. However, it is pretty clear that the Treasury Department, or the Markets area, plays a key role within the organization of the investment banking business of any bank entity.

The Treasury department is in close and constant communication with the rest of departments, both internally and externally. Therefore, all the decisions and actions taken by the Bank's treasury have a direct impact on the financial world. Besides, the Treasury department handles a great amount of data in order to manage operations, confirmations, payments and the procurement of products.

Nowadays, most of the bank entities manage their fixed data through several applications, each of them using different programming languages: COBOL, C++, Java... This directly causes losses in money and time whenever the entities implement an exchange of information between applications.

This final degree project focuses on the design of an application architecture able to manage all the fixed data needed by the Bank's treasury and on its integration into a service oriented architecture (SOA). The mentioned data grows constantly geared towards the consumers' needs.

In order to validate the proposed SOA, an application will be created showing how it is able to manage the most important fixed data used by the Bank's treasury:

- Counterparties, comprising:
  - General attributes
  - Agreements
  - Dictionary
  - Contacts
  - Confirmations
  - Regulatory data
  - Settlement instructions
- Internal structure, comprising:
  - Portfolios
  - Groups
  - Offices

- Market conventions, comprising:
  - Index
  - Currencies
  - Calendars
  - Geographics units

# Capítulo 1

# Introducción

## 1.1. Motivación

Para los bancos es muy importante que la tesorería funcione correctamente y con la menor pérdida de tiempo posible, para de esta manera poder realizar todas las operaciones y transacciones de la manera más eficiente.

Las entidades financieras de primer nivel están en constante actualización de sus sistemas, para mantenerse al día con las nuevas tecnologías, facilitar su propio trabajo y ofrecer mejores servicios a sus clientes.

Unas de las partes más importantes dentro de la tesorería y que se encuentra en constante crecimiento, son los sistemas de almacenamiento de los datos fijos utilizados. En la gran mayoría de los bancos, todos estos datos se encuentran divididos entre varias aplicaciones, cada una de ellas con distintos tipos de tecnologías, como COBOL, Java, C++...

El hecho de no tener todos los datos fijos centralizados dentro de la misma aplicación y con la misma tecnología, genera una gran pérdida de tiempo a la hora del funcionamiento de los aplicativos encargados de la contrataciones, liquidaciones, confirmaciones..., ya que tienen que estar pidiendo información para cada acción que quieran realizar, a distintos aplicativos. También esto conlleva una pérdida de dinero, ya que cada vez que se quiere realizar una actividad nueva se tiene que tener en cuenta cómo conseguir los datos fijos de cada uno de los gestores de los datos.

Cada vez que entra en vigor una nueva regulación, el hecho de tener los datos fijos separados en distintos aplicativos también supone una pérdida de dinero para las entidades financieras. Cada vez que necesitan actualizar su forma de manipular sus datos, esto produce que tengan que realizar varios proyectos, basados en distintas tecnologías, para poder actualizar todos sus gestores de datos fijos.

Es por ello que se hace necesaria la definición de una arquitectura uniforme de aplicativo, independiente de los lenguajes de alto nivel, fácilmente extensible y actualizable. De tal manera que, si en el futuro una aplicación necesita añadir una nueva funcionalidad o actualizar alguna de las funcionalidades ya existentes, no haya que modificar todos los demás aplicativos con los que esté conectada.

## 1.2. Objetivos

El objetivo del TFG es definir una arquitectura de un aplicativo que haga de repositorio global de todos los datos fijos de una entidad bancaria, en el área de tesorería.

Dado lo señalado en la Sección 1.1, se ha visto que la división de los datos fijos en varios aplicativos genera una serie de problemas que se podrían solventar si toda la información se gestionase desde el mismo aplicativo.

Además se implementará un ejemplo de aplicativo siguiendo la arquitectura propuesta. Los objetivos principales de la arquitectura propuesta y el software a desarrollar son los los siguientes:

- **Sin coste:** Aunque el proyecto a realizar quiere generar un repositorio de datos fijos que se integre con los demás aplicativos de un banco, se debe realizar con los medios gratuitos disponibles.
- **Fácilmente actualizable:** Como se ha comentado con anterioridad, es necesario que el programa generado sea de fácil actualización, para reducir los costes a futuro.

- **Comunicación con resto de aplicativos:** Dado que la aplicación necesitará estar en constante comunicación con el resto de aplicativos del banco, será necesario recrear una **arquitectura SOA**, en la cual incluiremos la aplicación. Esta es la arquitectura que se está usando actualmente en los bancos de primer nivel mundial.

- **Facilidad de Uso:** Dado que esta aplicación será manejada por los usuarios de los datos fijos, los cuales no están acostumbrados a los cambios de aplicativos, esta debe ser fácil de usar para ellos.

Para la solución final se realizará un repositorio con un único dato fijo, en este caso serán las unidades geográficas, que cumpla con todos los objetivos anteriores.

# Capítulo 2

# Visión General

En este capítulo se explicará de forma general en qué consisten las herramientas usadas para llevar a cabo la definición de la arquitectura y la implementación del proyecto, y el motivo por el cual se han elegido en lugar de alguna de las otras opciones disponibles.

## 2.1. PHP

Existen gran cantidad de lenguajes de programación, con los cuales se podría realizar este aplicativo, tales como Java, Python, .NET, C++...

Dado que la aplicación que se va a crear será un programa que va a ser utilizado por una gran cantidad de usuarios, se ha decidido generar una aplicación web. Al realizar una aplicación web, se evitan problemas de difusión e instalación de la aplicación, ya que con facilitar la dirección web de la aplicación ya podrían empezar a utilizarla.

Finalmente se ha decidido utilizar PHP ya que puede ser desplegado en la mayoría de los servidores web y en casi todos los sistemas operativos y plataformas sin coste alguno, facilitando su incorporación a las entidades financieras.


Figura 2.1: Logo PHP.

Además de estos motivos, también se ha decidido utilizar PHP debido a las siguientes características de las que dispone:

- Este lenguaje de programación, permite realizar aplicaciones web dinámicas con fácil acceso a información almacenada en bases de datos.
- Tiene una gran conectividad con el sistema de gestión de bases de datos MySQL.
- Se puede expandir su potencial incluyendo módulos.
- Es libre, por lo que se puede usar sin coste alguno.

Aunque uno de sus inconvenientes es que al ser un lenguaje interpretado, es relativamente más lento que lenguajes de bajo nivel. Debido a que la aplicación a realizar es un repositorio de datos, no se prevé a futuro que vaya a requerir scripts complejos que puedan ralentizar la aplicación, por lo que este no sería un gran inconveniente.

## 2.2. MySQL

Dentro de la gran variedad de gestores de bases de datos existentes, tales como Oracle, JDBC, MongoDB, MySQL..., de los que se podía elegir para realizar el proyecto, se ha decido usar MySQL.


Figura 2.2: Logo MySQL.

MySQL es Open Source, por lo cual cumple con los objetivos propuestos para la realización del proyecto. Además tiene muy buena compatibilidad con el lenguaje de programación elegido para realizar la aplicación, PHP.

Alguna de sus otras características importantes por las que se ha elegido son:

- Soporta una gran cantidad de datos, por lo que es perfecta para almacenar todos los datos fijos que puede tener una entidad bancaria.
- Funciona en múltiples plataformas, facilitando su integración en los sistemas de las entidades financieras.
- Ofrece gran seguridad, por lo que puede almacenar los datos sensibles de los que puede disponer una entidad bancaria, tales como datos fiscales o cuentas.

Una de las razones más importantes para su elección, es que es uno de los gestores de bases de datos más extendidos y tiene un amplio subconjunto del lenguaje SQL. Esto facilitará su expansión en caso de ser necesario y reducirá costes al contar probablemente con gente con conocimientos para gestionarla.

## 2.3. XAMPP

Se ha utilizado para la realización del proyecto el paquete XAMPP. Este paquete es el que se ha utilizado para lanzar la aplicación web por ser de software libre y contener todo lo necesario para ello.


Figura 2.3: Logo XAMPP.

El paquete XAMPP contiene:

- El servidor web Apache: Servidor de HTTP de código abierto multiplataforma.
- Intérprete de lenguaje PHP. Se ha explicado el motivo de su elección en la Sección 2.1.

- El gestor de base de datos MYSQL. Se ha explicado el motivo de su elección en la Sección 2.2.

## 2.4. SOA (Service Oriented Architecture)

Para este proyecto es necesario realizar el repositorio de datos fijos de tal manera que se pueda incorporar dentro de una arquitectura orientada a servicios (SOA). Esto es debido a que a día de hoy, las grandes empresas, en las que están incluidas las entidades bancarias, utilizan esta arquitectura para la comunicación entre sus aplicativos. Para entender un poco mejor esta arquitectura y el por qué las grandes empresas están usándola, en este apartado se va a explicar en qué consiste y sus principales características.

La arquitectura SOA, como su propio nombre indica, es un estilo de arquitectura que está basada en la orientación a servicios. La orientación a servicios implica una forma de pensar sobre cómo implantar los servicios de un negocio para que puedan comunicarse de forma fluida unos con otros. Un servicio es una actividad realizada dentro de un negocio; en el caso que nos compete de las entidades financieras, un ejemplo de servicio puede ser la realización de transferencias.

El objetivo principal de la arquitectura SOA es hacer que cada aplicativo dentro de una empresa sea capaz de presentar los servicios que ofrece de una manera adecuada, que permita usar sus capacidades de forma homogénea.

En una arquitectura SOA cada sistema debe brindar servicios a desconocidos, por lo que debe centrarse en las capacidades que ofrece y no en quién va a recibirlas. De esta forma se pueden crear funciones que sean utilizables por distintos clientes, únicamente conociendo la descripción del servicio para poder utilizarlo.

Por estos motivos, las arquitecturas SOA están formadas por servicios de aplicaciones débilmente acoplados y altamente interoperables. Estos servicios que ofrecen sus funcionalidades, pertenecientes a una empresa, suelen ser servicios web (Sección 2.4.2), los cuales se conectan unos con otros generalmente a través de un bus de datos, denominado ESB (Sección 2.4.3), por el cual fluirá la información entre ellos.

### 2.4.1. Beneficios

Existen una serie de beneficios por los cuales las empresas están adoptando esta arquitectura; en la Figura 2.4 se pueden apreciar los 5 principales:


Figura 2.4: Principales beneficios al utilizar arquitectura SOA.

- **Reutilización:** gracias a la arquitectura SOA es posible realizar nuevos procesos de negocio reutilizando funcionalidades de aplicativos ya existentes que han sido expuestos como servicios web. De esta forma se consigue ganar tiempo y reducir costes ante nuevos requerimientos de negocio, dado que reescribir una aplicación o parte de ella no sería la mejor opción a seguir.

- **Interoperabilidad:** en la arquitectura SOA se utilizan protocolos estándares que permiten que servicios y clientes se pongan de acuerdo sobre la manera en la que comunicarse, teniendo de esta forma un mecanismo de comunicación que elimina las limitaciones que se puedan generar al intentar comunicarse entre diferentes plataformas o diferentes lenguajes de programación utilizados en los aplicativos.

- **Escalabilidad:** en general, una arquitectura es más escalable si añadir una nueva unidad de capacidad tiene menor coste que en otras.

  Dentro de la arquitectura SOA, una nueva unidad de capacidad sería equivalente a agregar un nuevo servicio web dentro de un sistema, con un nuevo proceso de negocio, para que interactúe con otros servicios.

  Dado que los servicios web no tienen estado, es decir que no necesitan almacenar la información de la sesión cada vez que se les invoca, la complejidad que da agregar nuevos servicios web es bastante baja.

- **Flexibilidad:** dado que en la arquitectura SOA los distintos aplicativos que forman parte de ella están centrados en los servicios que ofrece, aunque alguno de estos aplicativos se modifique, no es necesario realizar ningún cambio en los demás aplicativos que se comunican con él, ya que no se modificará la forma en la que estos se comunican entre ellos.

- **Eficiencia en coste:** como en la arquitectura SOA se reutilizan las funcionalidades de otros aplicativos ya existentes, esto reduce de manera muy significativa los costes de acometer nuevos proyectos dentro de las empresas, ya que reduce la inversión en infraestructura y reduce la necesidad de adquirir nuevo software especializado.

### 2.4.2. Servicios Web

Un servicio web por sí mismo no tiene por qué pertenecer a una arquitectura SOA. Para que un servicio web se pueda considerar que forma parte de una arquitectura SOA debe ser capaz de cumplir con los beneficios expuestos anteriormente en la Sección 2.4.1.

Los servicios web, son servicios que se comunican entre aplicaciones utilizando un conjunto de protocolos y estándares. Distintas aplicaciones software creadas en distintos lenguajes de programación y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como internet.

La comunicación se caracteriza por el intercambio de mensajes XML y por ser independientes del protocolo de comunicación. Para lograr esta independencia, el mensaje se envuelve de manera apropiada para cada protocolo gracias al protocolo de transporte SOAP (Sección 2.4.2).

El lenguaje de Descripción de los Servicios Web (WSDL) (Sección 2.4.2), expresado en XML, describe cómo acceder al servicio, de qué funciones dispone, qué argumentos necesita y qué devuelve cada uno.

#### WSDL (Web Services Description Language)

WSDL es un formato del XML que describe la interfaz pública de los servicios web. En él se describe cuáles son los requisitos del protocolo y cómo interactuar con los servicios de los que dispone el servicio web. Las operaciones y los mensajes que soporta se describen de manera abstracta y se ligan después al protocolo concreto de red y al formato del mensaje.

WSDL es como un contrato entre el proveedor del servicio y el cliente, en el que el proveedor del servicio indica:

- Qué funciones se pueden llamar.
- Qué tipos de datos utilizan las funciones.
- Qué tipo de transporte se usará para el envío y recepción de los mensajes (en este proyecto se usará SOAP (Sección 2.4.2)).
- Cómo acceder a estos servicios.

**SOAP (Simple Object Access Protocol)**

SOAP es un protocolo simple para el intercambio de información estructurada en un entorno distribuido y descentralizado. Especifica cómo dos objetos en diferentes procesos pueden comunicarse a través del intercambio de XML.

Gracias a que SOAP usa XML tiene la ventaja de que las personas pueden entenderlo con facilidad, pero a la vez es un inconveniente porque los mensajes son más largos.

La estructura de un mensaje SOAP tiene las siguientes partes:

- **Envelope (obligatoria):** raíz de la estructura. Es la parte que identifica al mensaje SOAP como tal.
- **Header:** permite enviar información relativa a cómo procesar el mensaje.
- **Body (obligatoria):** contiene la información relativa a la llamada y la respuesta.
- **Fault:** contiene información sobre posibles errores que se hayan podido producir durante el proceso del mensaje y el envío.

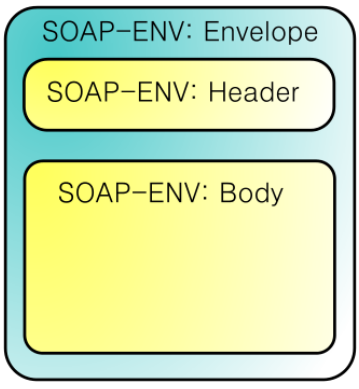


Figura 2.5: Estructura de un mensaje SOAP.

En la Figura 2.5 se puede ver cómo quedaría estructurado un mensaje SOAP con las principales partes Envelope, Header y Body complementadas.

### 2.4.3. ESB (Enterprise Service Bus)

Un ESB (en español Bus de Servicio Empresarial) es un modelo de arquitectura software que se encarga de gestionar las comunicaciones entre distintos aplicativos de una empresa. Es decir, es un punto central donde se gestionan todos los servicios expuestos por las aplicaciones de la empresa (sin importar en qué plataformas se encuentren) y sobre la cual se pueden construir aplicaciones que aprovechen las funcionalidades expuestas por los demás aplicativos.

Un bus de servicio empresarial proporciona una capa de abstracción que se construye sobre una implementación de un sistema de intercambio de mensajes de empresa. Esta capa de abstracción permite a los expertos en integración explotar el valor del envío de los mensajes sin tener que escribir código. En la Figura 2.6 se puede ver un ejemplo de cómo varios aplicativos de una empresa quedarían interconectados a través del ESB (mostrado como una tubería por la que circularía la información), compartiendo de esta forma la información entre todos ellos.

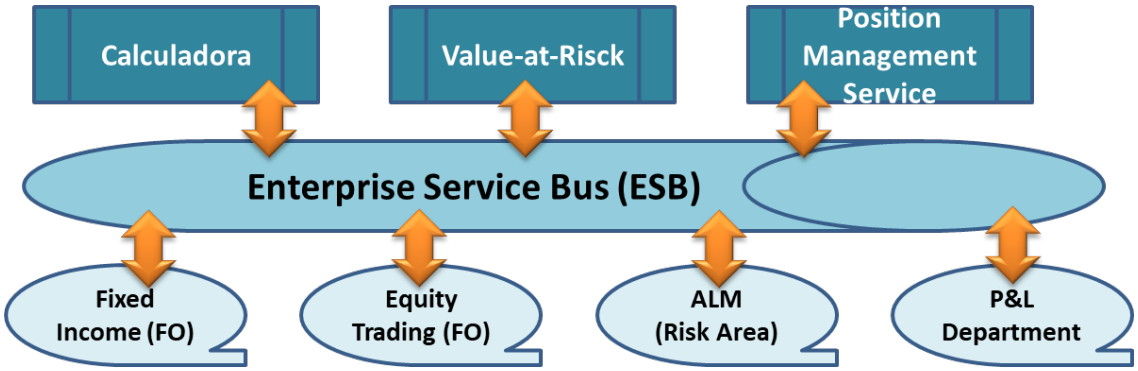


Figura 2.6: Ejemplo de un Enterprise Service Bus (ESB).

Un ESB no implementa por sí mismo una arquitectura SOA, sino que es una herramienta que proporciona las características necesarias para implementar una. El ESB trata de aislar el acoplamiento entre el servicio solicitado y el medio de transporte.

Algunas de las funciones más importantes que debe de tener un ESB son:

- **Invocación:** que sirva de soporte para protocolos de transporte síncrono y asíncrono.
- **Enrutamiento:** que sea capaz de enviar el mensaje al destino correcto basándose en normas, contenidos, políticas...
- **Mediación:** que contenga adaptadores, que sea capaz de realizar transformaciones de protocolos.
- **Transmisión de mensajes:** que procese los mensajes, los transforme y los mejore.
- **Coreografía de procesos:** que implemente procesos de empresa complejos.
- **Orquestador de servicios:** que coordine múltiples servicios presentados como un único servicio agregado.
- **Procesamiento de eventos complejos:** que interprete eventos, los correlacione y empareje patrones.
- **Otros servicios de calidad:** que sea seguro, con entregas confiables y que administre las transacciones.
- **Administración:** que sea capaz de monitorizar, auditar, registrar, mediar. Que tenga una consola de administración.

Además un ESB se debe caracterizar principalmente por:

- Ser capaz de interoperar entre aplicaciones creadas en distintos sistemas y con distintos lenguajes de programación.
- Uso general de XML.
- Facilidad para la transformación de formatos y valores de datos, entre el servicio emisor y el aplicativo receptor.
- Validación de esquemas para la emisión y la recepción en los aplicativos.

- División y combinación de múltiples mensajes.

- Encolamiento y retención de mensajes si las aplicaciones no están temporalmente accesibles.

### 2.4.4. WSO2

WSO2 es una compañía dedicada al desarrollo de aplicaciones open source enfocadas en proveer una arquitectura orientada a servicios.

Para este proyecto se ha decidido utilizar sus servicios dado que es open source, que es uno de los más potentes del mercado y que contiene un paquete de herramientas con el que poder llevar a cabo una arquitectura SOA de manera sencilla.

WSO2 se ha instaurado como una de las herramientas para el desarrollo de arquitecturas SOA más extendidas; por ejemplo se ha instaurado en estructuras IT tan complejas como Ebay, que lleva a cabo más de 1000 millones de transacciones diarias.

Algunos de los productos más destacados de la plataforma WSO2 son:

- **WSO2 API Manager:** es una solución completa que sirve para gestionar los servicios de publicación a terceros, garantizando la seguridad de la información y reduciendo los tiempos de integración.

- **WSO2 Identity Server:** es un producto que permite gestionar los protocolos y credenciales de acceso a los recursos y servicios corporativos a través de un único punto de gestión. WSO2-IS proporciona los protocolos de acceso más utilizados en el mercado, como puede ser OAuth2.

- **WSO2 Enterprise Service Bus:** es el ESB que proporciona WSO2. Es un producto de WSO2 que se encarga de gestionar la orquestación de servicios y accesos a recursos en los procesos de negocio. WSO2-ESB facilita la integración de todos los productos de WSO2.

- **WSO2 Data Analytics Server:** es un producto que complementa la solución WSO2 ayudando a gestionar el reporting de la información gestionada en la plataforma.

## 2.5. SoapUI

Para la realización de las pruebas sobre las aplicaciones con arquitectura orientada a servicios generadas, se va a utilizar la herramienta SoapUI.

Con esta herramienta se puede probar a realizar peticiones a los servicios web creados y ver qué respuestas se obtienen realizando pruebas de llamadas a las funcionalidades que ofrecen.

Alguna de sus principales funcionalidades son:

- Soporte a SOAP.
- Incorpora un monitor SOAP para capturar y analizar el tráfico.
- Genera de manera automática los tests y las peticiones SOAP de las operaciones definidas en el descriptor WSDL.

# Capítulo 3

# Plan de trabajo

Como plan de trabajo en el caso de este proyecto, tras haber estado investigando y haber elegido qué herramientas se van a usar para su desarrollo, antes de poder realizar el desarrollo de la solución, hay que aclarar qué especificaciones ha de seguir la arquitectura prototipo a implementar.

En este capítulo se especificarán los requisitos que debe tener la aplicación, y más concretamente el prototipo, que se van a desarrollar para este proyecto.

Para ello se explicará qué requisitos generales debe tener la aplicación y en el caso que sea necesario, qué requisitos específicos debe de tener el prototipo.

## 3.1. Aplicación Web

Como hemos visto en la Sección 2.1 finalmente se decidió por la utilización del lenguaje de programación PHP. Con este lenguaje se va a crear un prototipo de aplicación web que sea capaz de contener los datos fijos utilizados dentro de la tesorería bancaria, más específicamente, para este proyecto se va a implementar un prototipo de repositorio capaz de almacenar las unidades geográficas.

Hay varios requisitos generales a seguir para poder realizar este prototipo:

- Dado que la aplicación va a ser usada por los usuarios que se encargan de gestionar los datos fijos dentro de la tesorería, tiene que ser una aplicación de fácil manejo e intuitiva. Estos usuarios por lo general no son informáticos y ya tienen sus propias aplicaciones para manejar de forma separada los datos fijos, por lo que para facilitarles el cambio, esta nueva aplicación debe ser sencilla y contener todo lo que ellos necesiten.
- La aplicación debe tener un control de acceso de usuarios, ya que al ser datos sensibles no todo el mundo podrá acceder a los mismos para verlos o modificarlos.
- La aplicación debe poder actualizarse y añadir nuevos módulos de forma ágil y rápida. Esto se debe a que será una aplicación en constante crecimiento, añadiendo nuevos datos fijos y añadiendo nuevas reglas de negocio a los datos fijos ya existentes por regulaciones del mercado.

Para poder gestionar las unidades geográficas en específico, se deben cumplir los siguientes requisitos:

- Deben poder ser de distintos tipos, tales como ciudades, países, pueblos...
- Deben poder almacenar distintas traducciones. Esto es debido a que esta aplicación será el repositorio central de unidades geográficas del área de tesorería, por lo que debe ser capaz de almacenar los distintos tipos de nomenclaturas que vayan a ser usadas por los demás aplicativos.
- Debe tener un módulo en el que se puedan realizar nuevas altas de unidades geográficas.
- Debe tener un módulo en el que se puedan visualizar la unidades geográficas existentes.
- Debe tener un módulo en el que se puedan realizar modificaciones en las unidades geográficas existentes.

Esta aplicación se va a incorporar dentro de la arquitectura SOA del banco, por lo que con los requisitos anteriormente descritos se cumple con lo necesario en cuanto a ser una aplicación capaz de proveer sus servicios a cualquier otro aplicativo que lo necesite.

## 3.2. Base de datos

Para crear la base de datos, se ha decidido usar el gestor de base de datos MySQL, como se describió en la Sección 2.2. Con este gestor se va a generar una base de datos que sea capaz de almacenar todos los datos fijos del área de tesorería.

La base de datos que necesita el aplicativo tiene una serie de requisitos importantes generales que debe cumplir:

- Debe contener una tabla donde almacenar los usuarios que van a tener acceso a la aplicación. La gestión de los usuarios se llevará a cabo a través del soporte técnico que se quedará a cargo de la aplicación una vez entregada, por este motivo la gestión se realizará a través de la BBDD.
- Debe ser capaz de almacenar grandes cantidades de datos y estos deben poder relacionarse entre ellos, ya que muchos de ellos están compuestos por un subconjunto de los demás. Por poner un ejemplo, un contrato va a estar compuesto por otros datos fijos como:
  - Contrapartidas.
  - Unidades geográficas.
  - Divisas.
  - Oficinas.
  - Productos.
  - Contactos.
- Los datos almacenados no se podrán eliminar de forma física de la base de datos, únicamente se podrán eliminar de manera lógica.

En el caso particular del prototipo, debe cumplir los siguientes requisitos:

- Debe almacenar las unidades geográficas de forma que más adelante se puedan relacionar con otros datos fijos.
- Debe ser capaz de almacenar distintos tipos de unidades geográficas.
- Debe contener los diferentes identificadores que pueda tener una unidad geográfica.

## 3.3. Servicio Web

Dado que este aplicativo se va a incorporar a una arquitectura SOA y debe ofrecer sus servicios a otros aplicativos, este deberá contener un servicio web que facilite ciertas funcionalidades.

Los servicios que pueda ofrecer a otros aplicativos serán dependientes de lo que se vaya creando, pero aunque en este prototipo sólo se vaya a implementar la gestión de las unidades geográficas, todos los futuros datos fijos que se vayan incorporando, como mínimo, deben de tener las siguientes funcionalidades:

- Envío de toda la información almacenada de un tipo de dato fijo. El prototipo a generar debe poder enviar todas las unidades geográficas que estén almacenadas.
- Envío de un único dato. Debe poder enviar para un tipo de dato fijo un único dato elegido.
- Envío de todos los identificadores de un tipo de dato fijo. Debe poder enviar para un tipo de dato fijo todos los identificadores que estén almacenados, tanto internos como externos.
- Envío de identificadores de un tipo de dato fijo. Al prototipo en cuestión se le deberá poder realizar la petición de los identificadores de alguna de las siguientes maneras:
    - Por identificador interno del prototipo. De tal manera que envíe todos los identificadores externos.
    - Por identificador interno del prototipo y un aplicativo externo dado. De tal manera que envíe el identificador externo pedido.

- Por identificador externo de otro aplicativo. De tal manera que envíe el identificador interno del prototipo.

Poniendo un ejemplo, si tenemos Madrid almacenado como unidad geográfica, teniendo como identificador interno del prototipo MAD y como identificadores externos MADRID para el aplicativo externo A y M para el aplicativo externo B:

- Debe poder realizar la petición de todos los identificadores externos buscando por el identificador del prototipo MAD, devolviendo MADRID para el aplicativo externo A y M para el aplicativo externo B.
- Debe poder realizar la petición de un identificador externo buscando por el identificador del prototipo MAD y por ejemplo el aplicativo externo A, devolviendo MADRID.
- Debe poder realizar la petición del identificador interno del prototipo buscando por M del aplicativo externo B, devolviendo el identificador interno del prototipo MAD.

## 3.4. SOA

La aplicación creada debe incorporarse a un repositorio de aplicaciones para que cualquier aplicativo que pueda necesitar sus servicios la vea y pueda dar uso de las funcionalidades creadas por su servicio web.

En el caso del prototipo, este debe ser subido al repositorio de APIs de la herramienta para arquitectura SOA que se ha elegido para el desarrollo del proyecto WSO2, descrito en la Sección 2.4.4.

# Capítulo 4

# Solución

En este capítulo se explicará la forma en la que se ha realizado el proyecto teniendo en cuenta los requisitos anteriormente mencionados en el Capítulo 3.

## 4.1. Prototipo Aplicación Web

Para el prototipo de la aplicación web se ha decidido crear distintos módulos en PHP de tal forma que se cumpla con los requisitos propuestos en la Sección 3.1. El módulo principal que ejecutará el prototipo y que se encargará de redireccionar hacia los otros módulos según las opciones elegidas es el archivo *index.php* (se encuentra en el material adjunto con la entrega de esta memoria). Quedando finalmente el prototipo con las siguientes funcionalidades:

- El prototipo tiene una ventana de **login**, mostrada en la Figura 4.1, en la cual los usuarios deben meter sus datos para poder acceder. Esta ventana esta implementada en el archivo *login.php* (se encuentra en el material adjunto con la entrega de esta memoria). Como ya se especificó anteriormente en la Sección 3.2, el alta de usuarios se llevará a cabo desde el servicio técnico que quedará a cargo del aplicativo, por lo que dentro de este prototipo para poder acceder se ha generado el usuario admin con contraseña admin.


Figura 4.1: Pantalla para insertar usuario y contraseña para acceder a la aplicación.

- Una vez dentro de la aplicación se ve, como muestra la Figura 4.2, un menú de **cabecera** en el cual el usuario podrá elegir el dato fijo con el que quiere trabajar; actualmente para este prototipo sólo está la opción de las unidades geográficas. Este menú permanecerá constantemente en pantalla. También se han incluido dos opciones arriba a la derecha, una de ellas para entrar en la edición del perfil y otra para cerrar la sesión. Este menú se ha implementado en el archivo *header.php*.

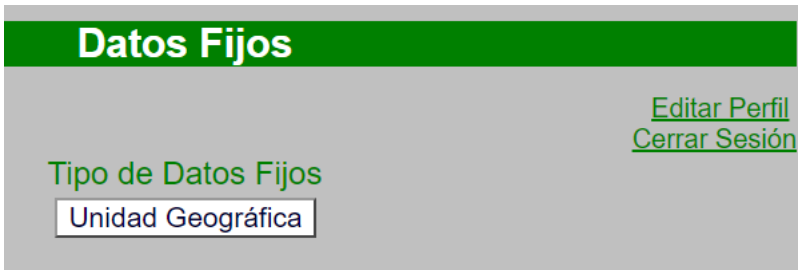


Figura 4.2: Pantalla de cabecera para seleccionar con qué dato fijo trabajar.

- Una vez seleccionado el tipo de dato con el que se quiere trabajar, aparece un menú, que se llamará a partir de ahora **menú geo unit**. Este menú, mostrado en la Figura 4.3, se ha implementado en el archivo *headerGeoUnit.php* con las siguientes opciones:
  - Búsqueda de una unidad geográfica, con la cual podremos localizar todas las unidades geográficas similares a lo que se inserte.
  - Un botón donde se podrá acceder a una nueva pantalla para realizar una nueva alta de una unidad geográfica.
  - Un botón con el que se mostrará el listado completo de unidades geográficas almacenadas.

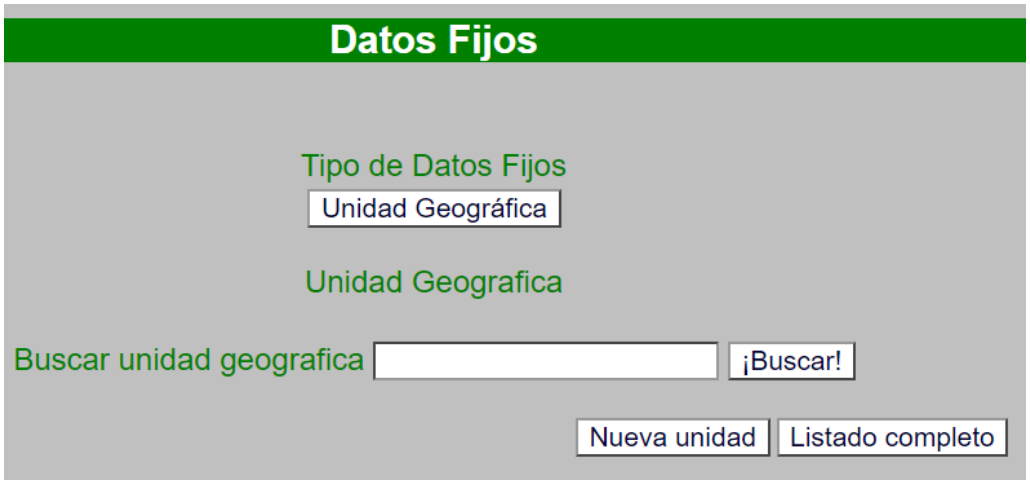


Figura 4.3: Pantalla desde la cual realizar búsquedas de Unidades Geográficas o dar de alta una nueva.

- Si se elige la opción de **buscar una unidad geográfica** desde el menú geo unit, se mostrará un listado con las unidades geográficas que tengan el identificador interno similar al que se ponga en el recuadro de búsqueda. Se llamará a esta pantalla a partir de ahora **menú consulta**. Esta pantalla, mostrada en la Figura 4.4, se ha implementado en el archivo *buscarGeoUnit.php*.


Figura 4.4: Pantalla que muestra las unidades geográficas encontradas; en este caso, buscando por ESP.

Como se puede ver en la Figura 4.4, una vez encontrada la unidad geográfica muestra sus valores: nombre de la unidad (su identificador interno), descripción de la unidad, tipo de unidad y estado. El estado se usará para saber si la unidad geográfica está operativa o ha sido eliminada.

A la derecha de la información de la unidad geográfica insertada existen 3 botones con los cuales se podrá trabajar con ella:

- El botón de consulta, con el cual se podrá acceder a la información de la unidad geográfica y a todos sus identificadores.
- El botón para edición, con el cual se podrá editar la unidad geográfica y sus identificadores y también se podrá añadir nuevos. Este botón sólo estará accesible si la unidad geográfica se encuentra activa.
- El botón de borrar, con el cual se realizará un borrado lógico de la unidad geográfica dejándola en estado inactivo. Este botón sólo estará accesible si la unidad geográfica se encuentra activa.

- Pulsando el **botón de consulta dentro del menú consulta**, se mostrarán los datos de la unidad geográfica. Este menú, mostrado en la Figura 4.5, se ha implementado en el archivo *consultarGeoUnit.php*.

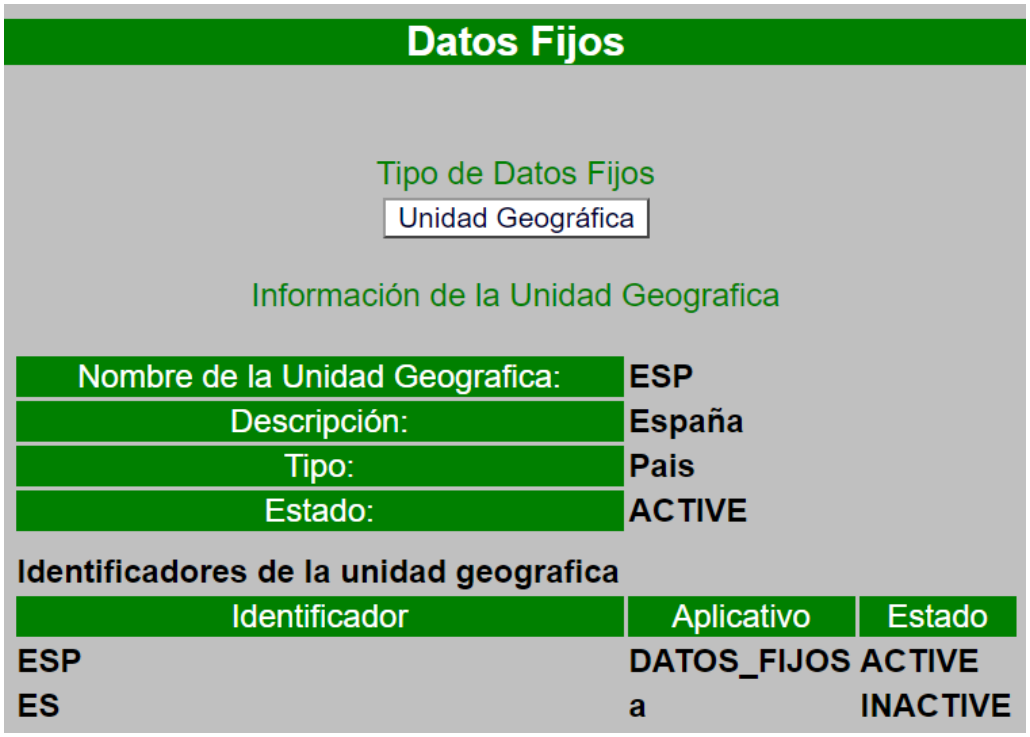


Figura 4.5: Pantalla de consulta de los datos de una unidad geográfica.

Como se puede apreciar en la Figura 4.5, se ve que la unidad geográfica tiene 2 identificadores, uno de ellos interno (el del aplicativo DATOS_FIJOS) y uno externo perteneciente en este caso al aplicativo a.

- Pulsando el **botón editar dentro del menú consulta**, aparecerá una pantalla con los datos de la unidad geográfica, en la que se podrá realizar modificaciones sobre la misma. Se llamará a esta pantalla a partir de ahora **menú edición**. Esta pantalla, mostrada en la Figura 4.6, se ha implementado en el archivo *geoUnit.php*.

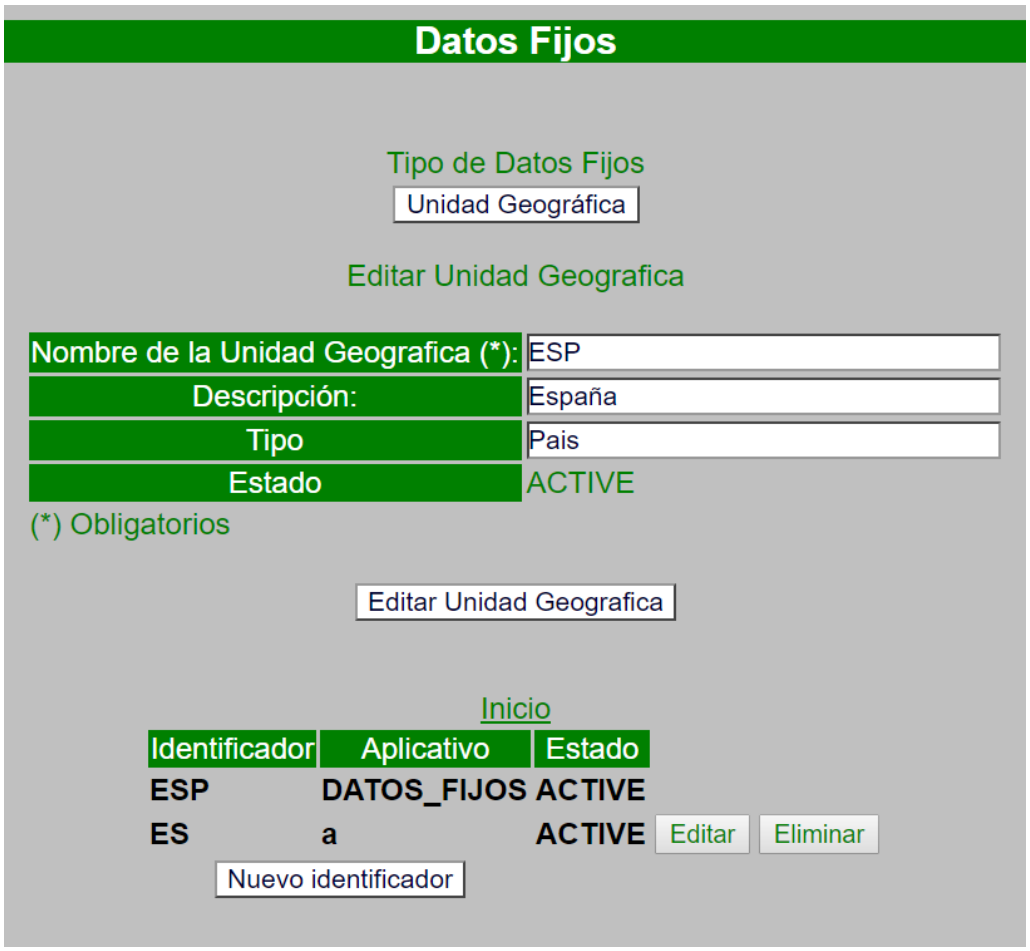


Figura 4.6: Pantalla para editar los datos de una unidad geográfica, incluyendo edición y alta de identificadores.

Una vez dentro del menú edición, como se puede observar en la Figura Figura 4.6, si se quiere modificar la información de la unidad geográfica sólo se tendrán que realizar los cambios necesarios en los recuadros accesibles y pulsar el botón de editar unidad geográfica.

En el caso de los identificadores, se podrá ver sus informaciones: identificador, aplicativo al que pertenece y su estado. A la derecha de la información de cada identificador aparecen dos botones para editar y eliminar, menos en el caso del identificador interno, el cual para modificarlo habría que modificar el nombre de la unidad geográfica y para eliminarlo, habría que eliminar la unidad geográfica.

Debajo del listado de identificadores hay un botón con el cual se pueden dar de alta nuevos identificadores.

- Accediendo a la **edición de un identificador desde el menú edición**, aparecerá un menú con el cual se podrá modificar los datos del mismo. Este menú, mostrado en la Figura 4.7, se ha implementado en el archivo *geoUnitId.php*.

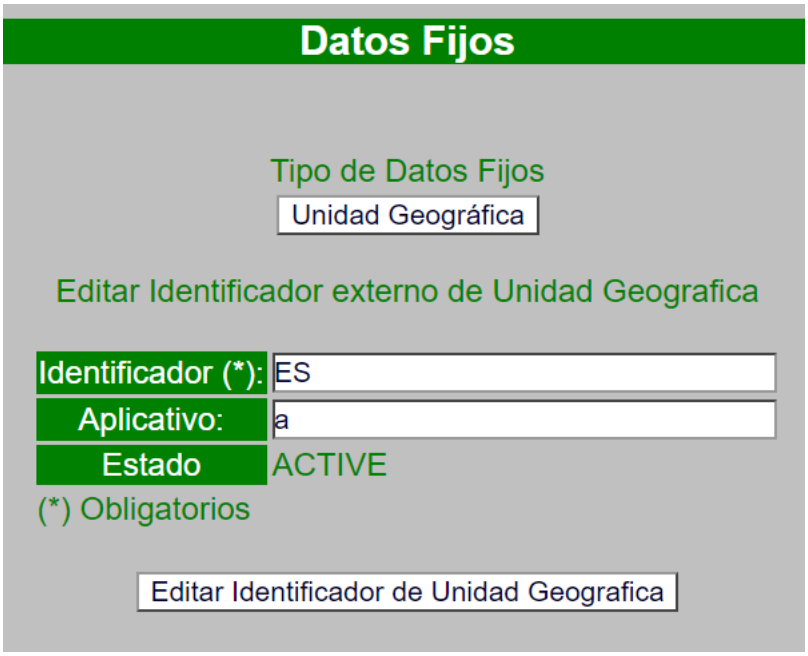


Figura 4.7: Pantalla de edición del identificador de una unidad geográfica.

Una vez dentro de la edición del identificador, como se puede observar en la Figura 4.7, si se quiere modificar la información sólo se tendrá que realizar los cambios necesarios en los recuadros accesibles y pulsar el botón de editar identificador de unidad geográfica.

- Pulsando el **botón de eliminar identificador dentro del menú edición**, se realizará un borrado lógico del mismo.

- Dentro del menú edición, si se pulsa sobre el botón para dar de **alta un nuevo identificador**, aparecerá una pantalla para realizar el alta. Esta pantalla, mostrada en la Figura 4.8, se ha implementado en el archivo *geoUnitId.php*.

  Sólo habrá que rellenar los campos y pulsar sobre el botón de alta. No se podrá dar de alta un identificador con un nombre para un aplicativo externo ya almacenado.

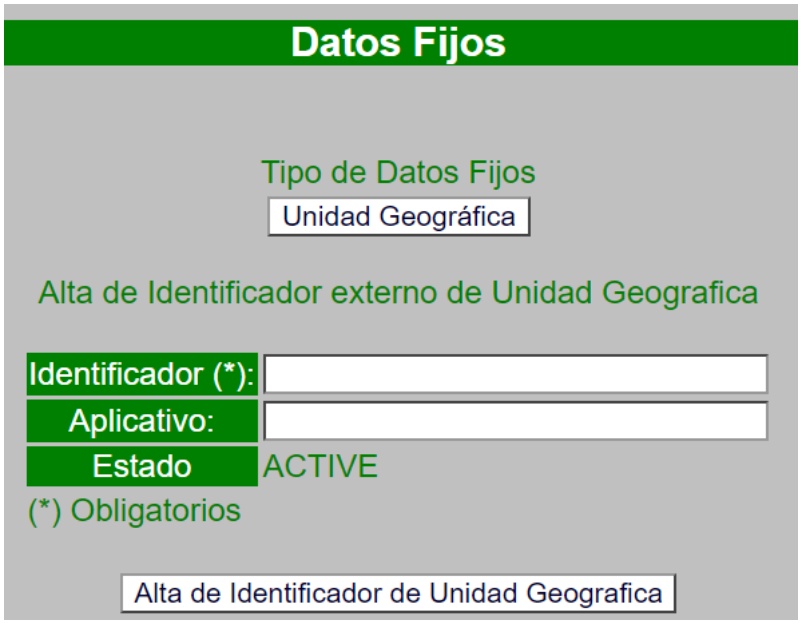


Figura 4.8: Pantalla de alta de un identificador para una unidad geográfica.

Como se puede observar en la Figura 4.8, por defecto, y al ser un alta, siempre se pondrá el estado como ACTIVE.

- Pulsando el **botón del listado completo desde el menú geo unit**, se mostrará un listado con todas las unidades geográficas almacenadas. Este listado contendrá la misma información para cada unidad geográfica y las mismas opciones sobre ellas que el menú consulta. Esta pantalla, mostrada en la Figura 4.9, se ha implementado en el archivo *buscarGeoUnit.php*.

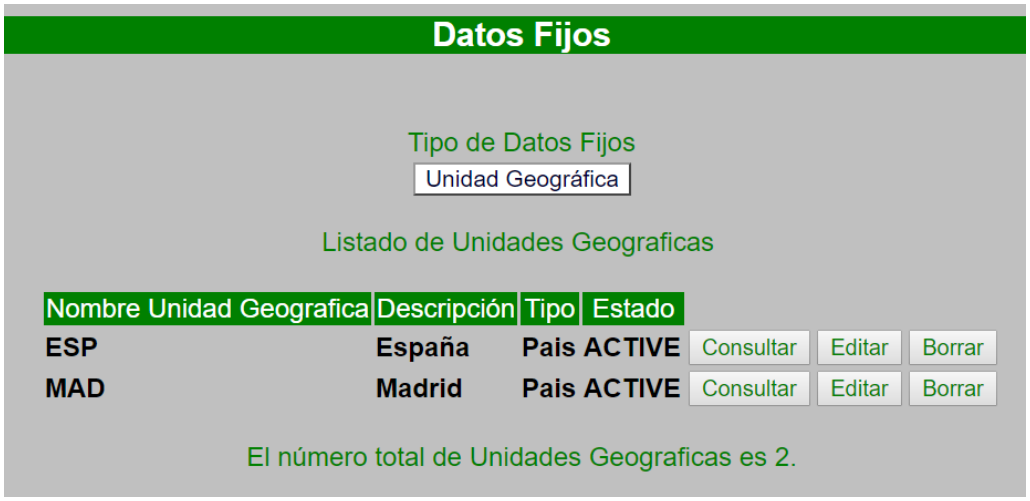


Figura 4.9: Pantalla con el listado completo de unidades geográficas.

- Pulsando sobre el **botón borrar de alguna de las unidades geográficas** desde el menú consulta, se realizará un borrado lógico de la unidad geográfica y sobre su identificador interno, inactivando los dos.

- Pulsando sobre el **botón de alta de unidad geográfica dentro del menú geo unit**, se accederá a una pantalla desde la cual se podrá realizar un alta de una unidad geográfica. Esta pantalla, mostrada en la Figura 4.10, se ha implementado en el archivo *geoUnit.php*.

  Para realizar el alta sólo habrá que rellenar los datos y pulsar sobre el botón de alta. Dando de alta una unidad geográfica nueva, también se dará de alta automáticamente su identificador interno. No se podrá dar de alta una unidad geográfica con un nombre ya almacenado.

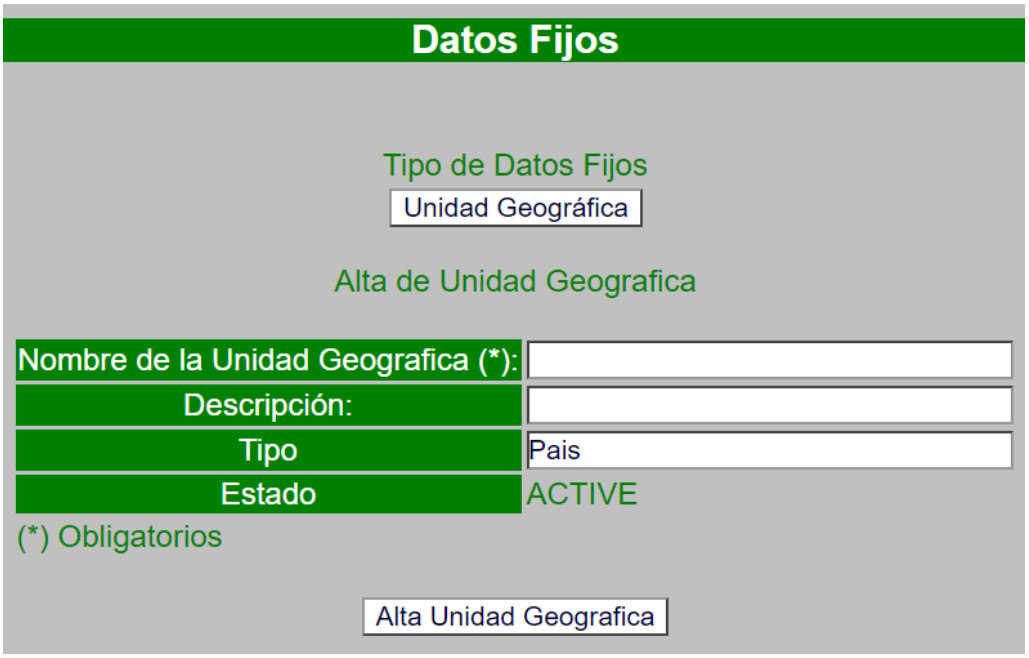


Figura 4.10: Pantalla de alta una nueva unidad geográfica.

Como se puede observar en la Figura 4.10, por defecto, y al ser un alta, siempre se pondrá el estado como ACTIVE.

- Pulsando sobre la opción de **editar perfil**, accesible desde todas las pantallas, se accede a un menú en el cual se podrá modificar los datos de usuario. Este menú, mostrado en la Figura 4.11, se ha implementado en el archivo *perfil.php*.

  Para realizar la modificación sólo habrá que modificar los datos y pulsar sobre el botón editar.


Figura 4.11: Pantalla para editar los datos de usuario.

Además de los archivos mencionados, también se ha creado una serie de archivos auxiliares para ayudar con las funcionalidades del prototipo:

- **Archivo *geoUnitFun.php*:** en este archivo se encuentran todas las llamadas a base de datos del prototipo, búsquedas, altas, modificaciones y selecciones relacionadas con las unidades geográficas.
- **Archivo *usuario.php*:** en este archivo se encuentran todas las llamadas a base de datos del prototipo, búsquedas y modificaciones relacionadas con los usuarios.
- **Archivo *funciones.php*:** en este archivo se encuentran dos funciones encargadas de evitar inyecciones SQL.
- **Archivo *footer.php*:** en este archivo se cierra el html del prototipo.

Con estos módulos el prototipo cumple con todos los requisitos generales propuestos para la aplicación siendo una aplicación de fácil manejo e intuitiva, con control de accesos y con facilidad de actualización, añadiendo nuevos módulos y únicamente teniendo que modificar de los archivos ya existentes el archivo *index.php*.

El prototipo también cumple con todos los requisitos específicos para el prototipo pudiendo almacenar distintos tipos de unidad geográfica, almacenando traducciones a través de los identificadores y pudiendo visualizar, modificar y realizar nuevas altas de unidades geográficas.

## 4.2. Prototipo Base de Datos

El prototipo de base de datos que se ha creado con MySQL, se compone de tres tablas con las que manejar la información de las unidades geográficas, sus identificadores y los usuarios.

- **Tabla geounit:** esta tabla es la que almacena las unidades geográficas. Está compuesta por:
  - *Id*: que es el identificador numérico único y privado para cada unidad geográfica. Este identificador se autoincrementará cada vez que se dé un alta nueva y no puede ser nulo. Es la clave primaria de la tabla.
  - *Nombre_geounit*: que es el campo que almacena el nombre e identificador público de la unidad geográfica. Este campo no puede ser nulo.
  - *Descripción*: que es el campo que almacena la descripción de la unidad geográfica.
  - *Tipo*: que es el campo encargado de almacenar el tipo de unidad geográfica almacenada. Este campo no puede ser nulo.
  - *Estado*: que es el campo encargado de almacenar el estado de la unidad geográfica. Cuando el dato se encuentre activo, este será ACTIVE y cuando se elimine el dato, se realizará el borrado lógico y pasará a ser INACTIVE. Si no se inserta este campo en el alta, se almacenará por defecto como ACTIVE.
- **Tabla geounitid:** esta tabla es la que almacena los identificadores de la unidad geográfica. Está compuesta por:
  - *Id*: que es el identificador numérico único y privado para cada identificador de unidad geográfica. Este identificador se autoincrementará cada vez que se dé un alta nueva y no puede ser nulo. Es la clave primaria de la tabla.

- *Geounit_id*: que es el campo que identifica a qué unidad geográfica pertenece el identificador. Este campo no puede ser nulo y es clave foránea con el campo id de la tabla geounit.
- *Valor*: que es el campo que almacena el valor del identificador, en el caso del identificador interno del prototipo para una unidad geográfica, será el mismo que el almacenado en el campo nombre_geounit de la tabla geounit.
- *Aplicativo*: que es el campo que almacena a qué aplicativo externo pertenece este identificador. En el caso del identificador interno del prototipo, el valor será DATOS_FIJOS.
- *Estado*: que es el campo encargado de almacenar el estado de la unidad geográfica. Cuando el dato se encuentre activo, este será ACTIVE y cuando se elimine el dato, se realizará el borrado lógico y pasará a ser INACTIVE. Si no se inserta este campo en el alta, se almacenará por defecto como ACTIVE.

- **Tabla usuarios:** esta tabla es la que almacena la información relativa a los usuarios del prototipo. Está compuesta por:
  - *DNI*: que es el campo encargado de almacenar el DNI del usuario. Este campo es la clave primaria de la tabla. El campo no puede ser nulo.
  - *Nombre*: que es el campo encargado de almacenar el nombre del usuario. Este campo no puede ser nulo.
  - *Apellidos*: que es el campo encargado de almacenar los apellidos del usuario. Este campo no puede ser nulo.
  - *Email*: que es el campo encargado de almacenar el email del usuario. Este campo no puede ser nulo.
  - *Usuario*: que es el campo que se usa como usuario para acceder al prototipo.
  - *Password*: que es el campo que almacena la contraseña de acceso al aplicativo. Ya que este campo será dado de alta por el servicio técnico que se quede a cargo de la aplicación, se debe cifrar con md5 y añadiendo al final de la contraseña hashSOA, para que de esta forma no se puedan ver las contraseñas por terceros. Tanto para el acceso como para la modificación de la contraseña se ha tenido en cuenta el cifrado.

- *Estado*: que es el campo encargado de almacenar el estado del usuario. Cuando el dato se encuentre activo, este será ACTIVE y cuando el usuario ya no vaya a utilizar el prototipo deberá realizarse el borrado lógico y pasar a ser INACTIVE. Si no se inserta este campo en el alta, se almacenará por defecto como ACTIVE.

Con estas tres tablas se cumple con los requisitos generales propuestos en la Sección 3.2, ya que los usuarios se almacenan de manera correcta y con contraseña privada y la base de datos será capaz de almacenar grandes cantidades de datos y a través de sus identificadores privados y únicos se podrán relacionar entre ellas, pudiendo así generar futuros datos compuestos.

Con estas tablas también se cumple con los requisitos específicos del prototipo, almacenando las unidades geográficas de tal manera que más adelante se puedan relacionar con otros datos fijos a través de su identificador privado, almacenando distintos tipos de unidades en el campo tipo de la tabla geounit y se almacenan los identificadores de aplicativos externos en la tabla geounitid.

Como se mencionó en la Sección 4.1, los accesos a base de datos se realizan desde los archivos *usuario.php* y *geoUnitFun.php*. Además existen otros archivos con información del prototipo de la base de datos y con los datos de acceso:

- **Archivo *config.php*:** este es el archivo donde se almacenan los datos de la base de datos con los que el prototipo accede a ella.

- **Archivo *inicio.php*:** este es el archivo con el cual el prototipo accede a la base de datos utilizando la información del archivo config.php.

- **Archivo *fin.php*:** este es el archivo que utiliza el prototipo para finalizar la sesión con la base de datos.

Adicionalmente en el archivo *BBDD.sql* se pueden ver los comandos usados para las creaciones de las tablas y el insert para agregar usuarios nuevos.

## 4.3. Prototipo Servicio Web

El prototipo de servicio web se ha creado usando el kit de herramientas NuSOAP. Gracias a este kit y utilizando las funciones generadas en el archivo *geoUnitFun.php*, se ha creado un servicio que cumple con los requisitos propuestos en la Sección 3.3 y ofrece las siguientes funcionalidades:

- A través de la función **getGeoUnit** devuelve la unidad geográfica pedida. Esta función tiene como parámetro de entrada el identificador interno del prototipo de la unidad geográfica y devuelve un json con todos los valores de la unidad geográfica de la tabla de la BBDD *geounit*.
- A través de la función **getAllGeoUnit** devuelve el listado completo de unidades geográficas. Esta función no tiene ningún parámetro de entrada y devuelve un json con todos los valores de todas las unidades geográficas dentro de la tabla de la BBDD *geounit*.
- A través de la función **getGeoUnitIdEx** devuelve el identificador interno del prototipo de una unidad geográfica. Esta función tiene como parámetros de entrada un identificador externo y el aplicativo externo al que pertenece, y nos devuelve los valores del identificador interno del prototipo de la tabla de la BBDD *geounitid*.
- A través de la función **getGeoUnitIdIn** devuelve el identificador externo de un aplicativo de una unidad geográfica. Esta función tiene como parámetros de entrada el identificador interno del prototipo y el aplicativo externo del cual se quiere saber su identificador, y nos devuelve los valores del identificador externo del aplicativo seleccionado de la tabla de la BBDD *geounitid*.
- A través de la función **getAllGeoUnitId** devuelve todos los identificadores pertenecientes a una unidad geográfica. Esta función tiene como parámetro de entrada una unidad geográfica y devuelve todos los valores de todos los identificadores pertenecientes a la unidad seleccionada de la tabla *geounitid*.

- A través de la función **getAllId** devuelve todos los identificadores de unidades geográficas del prototipo. Esta función no tiene parámetro de entrada y devuelve todos los valores de todos los identificadores de todas las unidades geográficas de la tabla *geounitid*.

Estas funciones se han implementado en el archivo *geoUnitService.php*.

Con estas funcionalidades, el prototipo de servicio web cumple con todos los requisitos propuestos: envía toda la información de un tipo de dato, envía un único dato elegido y envía los identificadores de un tipo de dato de las maneras especificadas, incluyendo una extra que envía todos los identificadores.

El prototipo de servicio web creado genera un archivo *geoUnitFun.wsdl* con toda la información de sus funcionalidades 2.4.2. Utilizando este archivo se ha podido probar a través de la herramienta SoapUI, explicada en la Sección 2.5, que el prototipo de servicio web funcionaba correctamente, gracias a los tests de prueba y realizando peticiones SOAP en xml como:

Listing 4.1: Petición getGeoUnitIdIn

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soapenv="http
    ://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:
    funcionesGeoUnit">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:getGeoUnitIdIn soapenv:encodingStyle="http://schemas.xmlsoap
          .org/soap/encoding/">
         <identificador xsi:type="xsd:string">ESP</identificador>
         <aplicativo xsi:type="xsd:string">a</aplicativo>
      </urn:getGeoUnitIdIn>
   </soapenv:Body>
</soapenv:Envelope>
```

Viendo como devuelve correctamente el xml:

Listing 4.2: Devolución getGeoUnitIdIn

```
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/
    soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
    envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-ENC="http://
    schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
      <ns1:getGeoUnitIdInResponse xmlns:ns1="urn:funcionesGeoUnit">
         <return xsi:type="xsd:string">[{"0":"3","Id":"3","1":"2","
            geounit_id":"2","2":"ES","valor":"ES","3":"a","aplicativo"
            :"a","4":"INACTIVE","estado":"ACTIVE"}]</return>
      </ns1:getGeoUnitIdInResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

## 4.4. Incorporación prototipo a SOA

Para incorporar el prototipo generado a la arquitectura SOA, en este caso representada por el set de herramientas WSO2, explicado en la Sección 2.4.4, se ha tenido que subir el servicio web del prototipo a la herramienta WSO2 API Publisher, para ello se ha tenido que utilizar la url que genera el archivo WSDL (http://localhost/SOA/src/geoUnitService.php?wsdl).

Una vez subido el servicio web al gestor de aplicaciones, cualquier otro aplicativo puede conectarse a él y realizarle peticiones SOAP gestionadas a través de la herramienta WSO2 Enterprise Service Bus, que hace las veces del ESB en esta arquitectura SOA.

Para comprobar que el prototipo de servicio web estaba correctamente subido, se ha utilizado la herramienta WSO2 API Store, desde la cual cualquier aplicativo se puede suscribir a las aplicaciones publicadas y realizarle peticiones de prueba.

Se ha usado para la prueba la misma petición SOAP usada en las pruebas del servicio web con la herramienta SoapUI, dentro de la Sección 4.3.

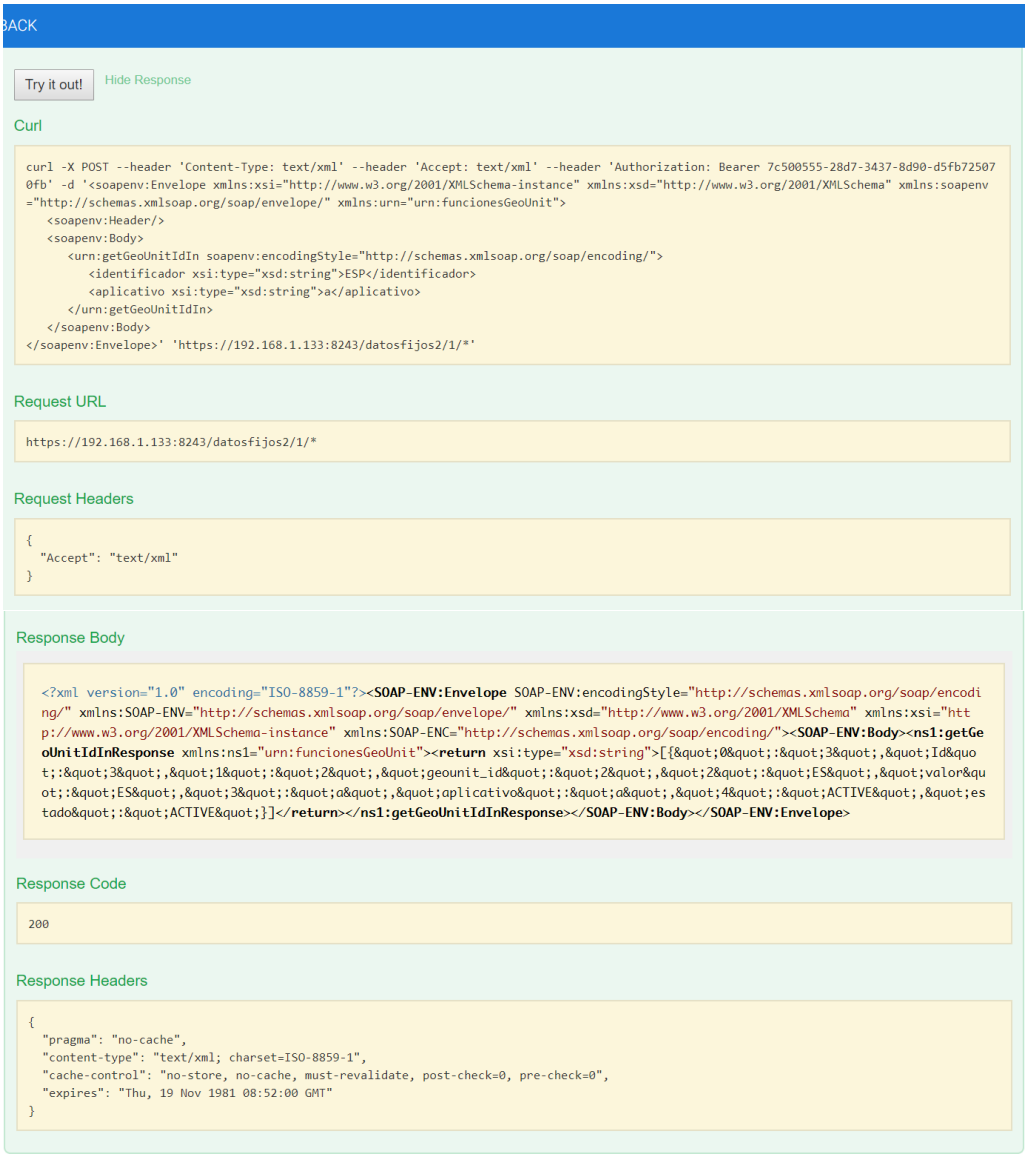


Figura 4.12: Datos de entrada y salida en la petición de prueba en WSO2.

En la Figura 4.12 se puede observar en la caja "*Curl*"la petición realizada al servicio web y en la caja "*Response Body*"la respuesta devuelta. Comparandola con la respuesta dada en la Sección 4.3, se ve que es correcta.

De esta forma queda comprobado que el servicio está integrado correctamente dentro de la arquitectura SOA y que cualquier aplicativo externo puede realizar peticiones al prototipo creado.

Una vez el aplicativo ha quedado integrado dentro de la arquitectura SOA, si necesita incorporar cualquier nueva funcionalidad o modificar alguna de sus funcionalidades ya existentes, sólo será necesario modificar el propio aplicativo y no será necesario modificar ninguna de la aplicaciones que estén conectadas a él.

Siendo una arquitectura de aplicativo implementada para incorporarse a la arquitectura SOA, se podrán ir incluyendo poco a poco los demás aplicativos de datos fijos existentes en las entidades financieras de forma sencilla, ya que es altamente actualizable y escalable, reduciendo así costes y tiempos a futuro.

# Conclusiones

## Conclusiones en Español

A través de los pasos que se han ido realizando en este proyecto, se ha visto cómo integrar un repositorio de datos fijos para la tesorería bancaria de una entidad financiera a una arquitectura SOA.

El prototipo final generado ha quedado bien integrado dentro de WSO2, la herramienta para arquitectura SOA elegida, y se ha desarrollado cumpliendo con las necesidades para poderse integrar correctamente con más aplicativos dentro de ella. Quedando de esta forma un prototipo de repositorio de datos fijos que a futuro se podría expandir con nuevos módulos de datos e incluirse dentro de una arquitectura SOA real de una entidad.

Tras la realización del prototipo final, se ha podido observar que la mayor dificultad de implementar aplicaciones para una arquitectura SOA dentro de una entidad, radica en el cambio que hay que realizar en la forma de pensar a la hora de crearlas. Para crear aplicaciones dentro de una arquitectura orientada a servicios hay que, además de pensar en las propias funcionalidades de la aplicación a crear, pensar en cómo ofrecer todos sus servicios a todos los aplicativos externos que puedan llegar a necesitarlos en el futuro, y no generar estos servicios para un sólo aplicativo externo.

En la actualidad, los aplicativos de las grandes empresas están desarrollados sin tener en cuenta la arquitectura orientada a servicios. Además, tienen una gran variedad de aplicativos desarrollados en diferentes lenguajes de programación y desplegados en distintas plataformas. Estos hechos suponen grandes pérdidas de tiempo y de dinero para las empresas cada vez que quieren realizar comunicaciones entre los diferentes aplicativos.

Este proyecto demuestra que la utilización de una arquitectura SOA en grandes empresas es una de las mejores maneras para intercomunicar las aplicaciones internas de una entidad, produciendo para estas un gran ahorro de tiempo y dinero al crear aplicaciones orientadas a ofrecer sus servicios a terceros y siendo estas altamente reutilizables y actualizables.

## Conclusions in English

Thanks to this project I have learnt how to integrate a fixed data repository into SOA at the Bank's Treasury at any financial entity.

The designed prototype is perfectly integrated into WSO2, the SOA toolkit chosen for this project. Besides, it has been developed facilitating its integration with more applications within this architecture. As a result, this fixed data repository is easily expanded with additional modules and could be included in a SOA of any current entity.

With the development of the prototype I realized that the most challenging aspect of implementing applications for SOA within an entity is the process-thinking of creating them. When creating applications for SOA it is important not only planning their own functions, but also offering a wide range of potential services useful for all the future external applications to be created – and not only for one single external application.

Nowadays, the fixed data of the big companies are developed without a SOA architecture. Besides, they present a great range of applications that use different programming languages and are displayed within varied platforms. As a result, the entities suffer losses in money and time whenever they implement an exchange of information between applications.

The purpose of this project is to show that the use of SOA is one of the best solutions to interconnect different internal applications within an entity. These applications are client oriented and easily reusable and updated; this yields important savings in time and money, especially for big companies.

# Bibliografía


[1] D. WOODS, T. MATTERNS, *"Enterprise SOA : Designing IT for Business Innovation"*, O'Reilly Media, 2006.

[2] J. BLOOMBERG, R. SCHMELZER, *"Service Orient or Be Doomed! : How Service Orientation Will Change Your Business"*, Wiley, 2006.

[3] E. NEWCOMER, G. LOMOW, *"Understanding SOA with Web Services"*, Addison-Wesley, 2005.

[4] THOMAS ERL, *"Service-Oriented Architecture. A field Guide to integrating XML and Web Services"*, Prentice Hall, 2004.

[5] G. ALONSO, F. CASATI, H. KUNO, V. MACHIRAJU, *"Web Services. Concepts, Architectures and Applications"*, Springer, 2004.

[6] YOAN ARLET CARRASCOSO PUEBLA, ENRIQUE CHAVIANO GÓMEZ, *"Arquitectura de software para el módulo de inventario del ERP cubano"*, Universidad de las Ciencias Informáticas, Cuba, 2008.

[7] DONNIE ROCK, *"Crear un webservice básico con PHP y SOAP"* : https://donnierock.com/2013/01/17/crear-un-webservice-basico-con-php-y-soap/, DonnierocK, 2013.

[8] ETHAN CERAMI, *"Web Services Essentials: distributed applications with XML-RPG, SOAP, UDDI & WSDL."*, O'Reilly Media, 2002.

[9] ERIC NEWCOMER, *"Understanding Web Services: XML, WSDL, SOAP AND UDDI"*, Addison-Wesley, 2002.

[10] JAMES SNELL, DOUG TIDWELL, PAVEL KULCHENKO, *"Programming Web Services with SOAP"*, O'Reilly Media, 2001.

[11] DAVID A. CHAPPELL, *"Enterprise Service Bus"*, O'Reilly Media, 2004.

[12] WSO2 API MANAGER, *"WSO2 API Manager Documentation"* : https://docs.wso2.com/display/AM210/WSO2+API+Manager+Documentation/

[13] WSO2 ENTERPRISE SERVICE BUS, *"WSO2 Enterprise Service Bus Documentation"* : https://docs.wso2.com/display/ESB500/WSO2+Enterprise+Service+Bus+Documentation/

# Agradecimientos

Gracias a José Luis Risco Martín del departamento de Arquitectura de Computadores y Automática por su colaboración y ayuda a la hora de realizar el proyecto.

# Autorización de difusión

## Autorización para la difusión del Trabajo Fin de Grado y su depósito en el Repositorio Institucional E-Prints Complutense

Los abajo firmantes, alumno y tutor del Trabajo Fin de Grado (TFG) en Ingeniería de Computadores de la Facultad de Informática, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor, el Trabajo Fin de Grado (TFG) cuyos datos se detallan a continuación. Así mismo autorizan a la Universidad Complutense de Madrid a que sea depositado en acceso abierto en el repositorio institucional con el objeto de incrementar la difusión, uso e impacto del TFG en Internet y garantizar su preservación y acceso a largo plazo.

TÍTULO del TFG: **Inclusión de un repositorio de datos fijos en un sistema de tesorería bancaria**.

Curso académico: 2016/2017

Nombre del Alumno: **Óscar López-Laguna Ortiz**.

Tutor del TFG: **José Luis Risco Martín**, Departamento de Arquitectura de Computadores y Automática.

Tutor del TFG: **Josué Pagán Ortiz**, Departamento de Arquitectura de Computadores y Automática.

| Firma del alumno | Firma del primer tutor | Firma del segundo tutor |
|---|---|---|