

# **Centralización de la Lógica de Negocio Mediante Web Services**

**Administración de Sistemas  
Informáticos en Red**

**Fco Daniel Ventura Serrano**

**I.E.S. Medina Azahara**

**14 de Junio de 2023**

## Índice

1. Título.....	2
2. Antecedentes .....	2
3. Definición del Problema .....	2
4. Justificación .....	3
5. Objetivos .....	3
Objetivo general .....	3
Objetivos específicos .....	3
6. Presupuesto .....	4
7. Limitaciones .....	5
8. Alcance .....	6
9. Marco Teórico .....	6
9.1. API's .....	6
9.2. SOAP vs REST .....	8
9.3. Windows Communication Foundation .....	8
9.4. Web Services .....	9
9.5. IIS Server y balanceo de carga .....	10
9.6. Firewall .....	11
10. Procedimientos .....	11
11. Descripción de las Actividades .....	12
11.1. Identificar fuentes de datos y necesidades .....	12
11.2. Definir DAO y DTO .....	12
11.3. Crear interfaces y servicios del Web Service .....	13
11.4. Desplegar Web Service en IIS y configurar balanceo de carga .....	14
11.5. Configurar firewall .....	14
12. Observaciones Finales .....	15
12.1. Posibles mejoras .....	15
13. Cronograma .....	15
14. Webgrafía .....	16

## 1. Título

---

Centralización de la Lógica de Negocio Mediante Web Services

## 2. Antecedentes

---

Las API son un ente que, desde el inicio del interés de cualquier persona en el mundo de los sistemas y/o el desarrollo, está presente, aunque en primera instancia no se tenga muy clara su importancia o por qué todo tiene una; cualquier red social, cualquier componente que incluyen las cuentas de google, esa aplicación en desuso que contabiliza el ejercicio que haces, la aplicación con la que te organizas las tareas o el calendario, y la lista sigue de forma interminable.

Todo necesita una API, y en cuanto llegas a un punto en el que necesitas una para solucionar un problema, sobre todo en un entorno profesional, entiendes de inmediato su importancia. Una API (application programming interface) es cualquier solución que permita comunicar diferentes aplicaciones o servicios, una interfaz entre máquinas.

De igual forma que una interfaz de usuario define qué datos recibe, qué acciones puede realizar y cómo un usuario, una API permite que dos sistemas se comuniquen, definiendo los datos que se pueden obtener, su formato, qué acciones se pueden disparar y en qué condiciones.

En un pequeño despliegue de aplicación o servicio tenemos que lidiar con este proceso y nos damos cuenta de la importancia de un buen uso de este tipo de interfaces, pero en un entorno profesional, cómo se implementen las API define la lógica de negocio y la optimización de todos los servicios y aplicaciones que se usan u ofrecen, es así que la curiosidad por esta parte tan fundamental me ha suscitado interés para desarrollar el proyecto sobre cómo utilizar una API para centralizar toda la información y lógica de negocio de una empresa.

Esta API puede contener partes sensibles o privadas, por lo que habrá que securizar cómo se muestra y a quién, cómo se accede a ella, etc. Con esta misión tendremos que definir muy bien la estructura de red interna de la organización y qué se expone al exterior.

## 3. Definición del Problema

---

Ya adelantamos en los antecedentes que las API son imprescindibles, pero cuando una organización ofrece varios servicios o productos todos relacionados de alguna manera, por ejemplo los TPV en tiendas, la gestión de inventario y pedidos de almacenes, páginas web, seguimiento de transporte, trazabilidad del producto, entre otras; encontramos que si estos diferentes servicios no se comunican de forma coherente, manteniendo integridad en los datos que manejan, la actividad no se puede llevar a cabo, incluso ocasionando problemas legales si entramos en el tratamiento de datos personales o informes relativos a la actividad empresarial.

Imaginemos que cada servicio o aplicación tuviese que definir cómo acceder a las propias bases de datos, gestionar la concurrencia, validar los datos, etc, esto sería un pozo en el que tirar el tiempo de la gente y una incubadora de errores e incongruencias por muy bien que se desarrollen e implementen todos los servicios, no se podría hacer un cambio sin tener que hacerlo en 20 sitios más, así como no se podría implementar un servicio nuevo sin estudiar todos los demás y “empezar de cero”, teniendo que volver a definir hasta lo más básico. Este planteamiento choca frontalmente con la lógica de los sistemas informáticos, donde las configuraciones, los métodos y procesos, los datos; todo se define en un único lugar al que acceden las partes del sistema que lo necesitan, logrando así una escalabilidad y rendimiento como la que se espera de cualquier sistema actual.

## 4. Justificación

---

Por otro lado imaginemos que creamos una abstracción de toda esa lógica, a la que realmente acceden nuestros servicios y aplicaciones, si hay cambios se realizarían solo en este paso intermedio o interfaz, además de que nuevas implementaciones podrían servirse de las partes de la interfaz que ya existe, añadiendo nuevas funcionalidades si las necesitase.

De esta manera sería mucho más viable organizar tanto los sistemas, como la labor humana, las desarrolladoras y administradoras sólo tendrían que conocer cómo comunicarse con esta interfaz para desempeñar su labor, separando el mantenimiento y configuración de los datos y la lógica de negocio de la operabilidad de los servicios y aplicaciones, pudiendo dirigir mucho más eficientemente los esfuerzos de las personas, separando los posibles errores, para así poderse subsanar mucho más rápidamente.

Abordando el tema de posibles errores es sencillo darse cuenta que esta centralización obviamente convierte al punto intermedio en un factor muy crítico para la actividad de la organización, es por ello que debemos contemplar las réplicas, el balance de carga y la securización de esta parte de nuestra organización.

Del mismo modo, las personas responsables de este desarrollo y despliegue tienen que tener altas capacidades para documentar, comunicar e instruir al resto de personal, ya que todos los servicios, los datos, y la seguridad, operan transversalmente sobre este sistema.

## 5. Objetivos

---

### Objetivo general

Se pretende lograr una abstracción de la lógica de negocio de los datos y el desarrollo, logrando escalabilidad y optimización de recursos para el desarrollo y mantenimiento de las distintas partes que participan en el aspecto tecnológico de la organización.

### Objetivos específicos

Identificar las necesidades de la lógica de negocio, definir los métodos para interactuar con los datos, así como los objetos que se manejan para dicha tarea, centralizar la forma y punto con el que acceder a ellos implementando una API SOAP Web Service de WCF, lograr una alta disponibilidad al servicio que cumple esta misión con balanceo de carga en el servidor y securizar su acceso.

## 6. Presupuesto

---

La inversión necesaria para llevar a cabo este proyecto se centra en un/os servidor/es con una potencia para cubrir la proyección de la actividad de nuestra organización, así como de expertos que puedan desarrollar y mantener un servicio tan neural en la actividad de la organización.

Nuestra solución con Web Service de WCF sigue la metodología SOAP para la API, lo cual requiere más recursos que las API REST, pero a cambio podemos manejar la securización con certificados y credenciales encriptadas, además de objetos mucho más fuertemente definidos y complejos. En nuestro presupuesto esto hace que necesitemos CPU multinúcleo, gran cantidad de RAM y los estándares más altos de velocidad de red si esperamos alto nivel de peticiones a la API.

Propuesta de servidor: hpe-proliant-dl380-gen10-intel-xeon-silver-4208-32gb, (~2500€) este equipo consta de 8 núcleos y 16 hilos, con hasta un gigabyte de memoria caché, 32 GB ampliables, además de algo muy importante, cuatro interfaces de red, pudiendo tener redundancia tanto el la IP que se expone, como en la conexión interna para hacer un cluster de servidores, con lo que haremos el balanceo de carga del Servidor IIS, en el que desplegaremos el WS.

El desarrollo se plantea entre cuatro y seis meses, implicando al menos a un desarrollador para gestionar el proyecto y codificar, tanto el código de los DAO, como del propio WS (6000~9000€). Continuando con los recursos humanos, en cuanto haya más de cinco servicios distintos que necesite cubrir la API, debería de haber una persona responsable para cada aplicación o servicio que vaya a consumir el WS, al menos en el inicio del planteamiento, estos puestos ya deberían de existir en la organización, por lo que he decidido no contabilizar estas tareas. También estaría involucrada en el proyecto, al menos una persona responsable de la red y los sistemas, para organizar el despliegue de servidores y la correcta implementación en la red y su consecuente securización, esta tarea no se expandirá durante todo el periodo del proyecto, sino más bien en sus pasos finales, digamos que alrededor de 1 mes (1500~3000€).

En resumen hablamos de cifras que oscilan alrededor de los 12000~17000€ contando con un balanceo de carga de dos servidores, aunque sería mucho más barato la ampliación en un futuro, ya que solo tendríamos que replicar otro servidor e incluirlo en el cluster de servidores.

En cuanto al mantenimiento, necesitaríamos de puestos de trabajo que, al menos parcialmente, se ocupen del correcto funcionamiento y posibles optimizaciones que surjan con el transcurso del tiempo y la evolución de la actividad de la organización. Si el servicio WS tiene que responder para una gran cantidad de servicios y aplicaciones, sería aconsejable incluso dedicar un departamento a la continua asesoría y mantenimiento de Web Service, así como de los DAO.

Cabe destacar que, tangencialmente a este proyecto se deben contar con equipamiento que permita su funcionamiento óptimo, como SAI, Switches de alto rendimiento, así como el acondicionamiento y cableado de la instalación, aunque son aspectos que se alejan de la propuesta y recaería sobre las particularidades logísticas de la organización que fuese a implementar el WS.

Para nuestra implementación, podemos funcionar con la versión gratuita de Visual Studio. Sin embargo, si prevemos la necesidad de soporte técnico, o una monitorización más compleja en la depuración, existen dos opciones de pago; la opción Standard, que incluye Azure Devops (para integración con la nube de Microsoft) y Visual Studio Professional IDE, tiene un costo de 45 USD al mes, que al cambio actual serían aproximadamente 42 euros. La opción Enterprise, por su parte, que incluye Azure Devops (Plan Básico + Test Plan) y Visual Studio Enterprise IDE, tiene un costo de 250 USD al mes, que al cambio actual serían aproximadamente 232 euros.

Concepto	Coste estimado
Servidor HP Proliant	2500€
Sueldo para desarrolladora (4~6 meses)	6000~9000€
Sueldo para administradora de sistemas (1~2 meses)	1500~3000€
TOTAL	12000~17000€

## 7. Limitaciones

Las limitaciones inmediatas radican en que nuestra propuesta está fundamentada en el conjunto de propuesta de una empresa privada como es Microsoft, obviamente nos aseguramos de una documentación profesional y unas garantías de uno de las grandes tecnológicas a nivel mundial, tanto en integración como comodidades, pero todo esto viene de la mano de tener que operar dentro del marco planteado por dicha empresa, cualquier función no contemplada por Microsoft será altamente compleja de implementar y rompería con las ventajas que conlleva asociarse con su completa propuesta de soluciones profesionales.

Aspectos más concretos de estas limitaciones serían, por ejemplo, para sacar partido de todo esto obviamente tenemos que desarrollar con el IDE Visual Studio, trabajando con .NET, tecnología que abstrae diferentes lenguajes de programación y sistemas operativos para tener unos objetos predefinidos comunes para todas las plataformas, lo cual adelanta en gran medida mucha parte del desarrollo, pero es un mundo turbulento y con cambios, por lo que es difícil administrar las versiones entre proyectos y más aún plantear interacciones con los diferentes sistemas operativos de forma más directa, dada su naturaleza de abstracción.

El propio Web Service que opera como API SOAP, ha de ser desarrollado usando ASP.NET, lo que conlleva que para desplegarlo tengamos que usar un equipo Windows Server con un IIS. Además, aunque desde .NET podemos acceder a datos de casi cualquier naturaleza que se nos ocurra, obviamente, la implementación para acceder a SQL Server es más versátil que para acceder a bases de datos de Oracle por ejemplo.

Tanto para los repositorios como para acceder al Windows server vamos a necesitar un sistema de usuarios de Windows, por lo que la opción más viable para manejar los usuarios de nuestra organización sería Active Directory de Windows Server.

Como vemos el compromiso con Microsoft se acaba extendiendo a la mayoría de aspectos principales de la actividad y gestión de la organización.

También tenemos que sopesar que al ser la solución de Microsoft, la norma hegemónica en el entorno profesional, los cibercriminales tienen sus esfuerzos focalizados en estos sistemas, por lo que necesitamos expertos en ciberseguridad que estén al día (como en cualquier otra solución, necesitamos una ciberseguridad fiable y robusta, pero el punto es qué la solución de Microsoft tiene un índice de amenaza muchísimo mayor).

La centralización que nos da el Web Service como API, también implica que el propio planteamiento de este servicio limita el resto de servicios de nuestra organización que accedan a los datos.

## 8. Alcance

---

Este proyecto desea cubrir las necesidades de una organización a la hora de organizar y optimizar el acceso a sus datos, adaptando y centralizando toda la lógica de negocio.

Una pequeña empresa posiblemente no necesite abordar esta necesidad mediante este proyecto, quizá le baste con usar soluciones de terceros tanto para definir cómo acceder a sus datos, así como con qué aplicaciones, ejemplos de este tipo de soluciones podrían ser el uso de software como factusol y toda la familia de programas para gestionar desde facturas, hasta CRM, pasando por TPV. A una escala mucho más grande y personalizada podemos ver soluciones como las que propone KAIS u Oracle, donde la estructura de los datos la marca la empresa según su lógica de negocio, pero el software para interactuar con ellas se compone de aplicaciones planteadas por esas empresas que se adaptan al caso concreto.

Este proyecto es esencial para empresas que necesiten muchas aplicaciones o servicios distintos para interactuar con los datos; por ejemplo una aplicación para que los transportistas aporten ubicación y estado de envíos, los operarios de almacén tengan aplicaciones para gestionar recepciones de materiales y la preparación de pedidos, todo esto teniendo que ser accedido de forma similar por los pedidos de una tienda online, etc. En este tipo de casos ya no hay una solución de terceros que nos ayude a gestionar nuestros servicios y aplicaciones para acceder a los datos, así que aunque se siga optando por soluciones de terceros, tendremos que empezar a desarrollar soluciones concretas y propietarias. Por muy pocas que se planteen en un inicio, desarrollar una API como se propone en este proyecto, prepara el camino para una optimización de recursos al desarrollar soluciones propias y concretas, así como la posibilidad de ir dependiendo menos de terceros.

Aplicar nuestra propias soluciones es complejo y un desafío, pero si se dispone de los profesionales adecuados, se traduce no solo en recortar costes, si no en control y análisis sobre la actividad empresarial y todos sus entresijos.

Estos motivos son suficientes para desde el primer momento desarrollar este proyecto en nuestra empresa y desplegar solo lo que analicemos que necesitemos, en lugar de adaptarnos a una opciones para soluciones poco personalizadas o con planes de pago inflados.

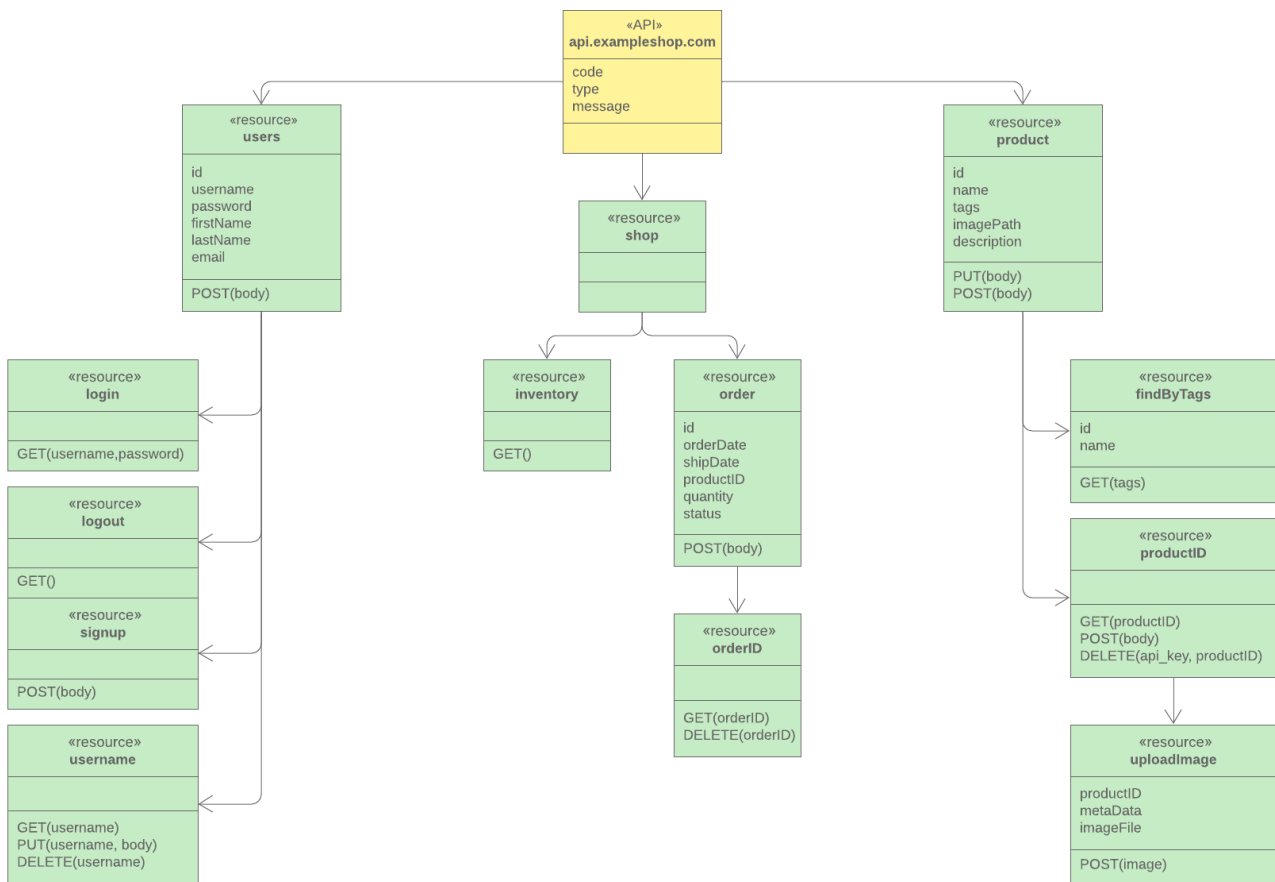
## 9. Marco Teórico

---

### 9.1. API's

Una API, o Interfaz de Programación de Aplicaciones, es un conjunto de reglas y protocolos que permite que diferentes programas de software interactúen entre sí. Funciona como un punto de encuentro, donde se define cómo se deben comunicar y compartir información dos o más sistemas. En esencia, una API proporciona un lenguaje común que permite que los sistemas colaboren y cooperen.

Al planificar el desarrollo de una API, es vital considerar el uso del Lenguaje Unificado de Modelado (UML). UML es una herramienta de modelado visual que describe y diseña sistemas de software. Ofrece varios tipos de diagramas, cada uno destinado a representar diferentes aspectos del sistema, desde su estructura general hasta los flujos de trabajo detallados y las interacciones entre los componentes. UML ayuda a visualizar y comprender la estructura de la API, identificar y diseñar las diferentes partes de la API y cómo interactúan entre sí. También es una herramienta eficaz para comunicar el diseño y la arquitectura de la API a otros miembros del equipo de desarrollo y documentar el diseño de la API para referencia futura.



- **Diagramas de clases:** muestran las clases del sistema, sus atributos y métodos, y las relaciones entre ellas.
- **Diagramas de casos de uso:** muestran cómo los usuarios o sistemas externos interactúan con el sistema que se está diseñando.
- **Diagramas de secuencia:** muestran cómo los objetos y componentes del sistema interactúan entre sí para realizar una función o proceso específico.
- **Diagramas de actividad:** representan el flujo de trabajo entre varios componentes del sistema.
- **Diagramas de estados:** muestran cómo cambia el estado de un objeto o componente a lo largo del tiempo.
- **Diagramas de despliegue:** muestran cómo se implementará el software en hardware.

Antes de embarcarse en la implementación de una API, es esencial analizar las necesidades del proyecto. No todas las APIs se crean de la misma manera y las necesidades específicas del proyecto pueden determinar la arquitectura, los protocolos y las tecnologías utilizadas en la API. Un análisis cuidadoso de las necesidades del proyecto puede ayudar a seleccionar la mejor API para el trabajo, asegurando que se satisfagan los requisitos de rendimiento, seguridad, escalabilidad y facilidad de uso.

Es por ello que una máxima a seguir para planificar una API, todos los departamentos y desarrollos de la organización tienen que ser partícipes, para poder cumplir las necesidades de todos los puntos de vista y conseguir la solución más homogénea posible, como cualquier buen desarrollo, los componentes tiene que ser factorizados, abstraídos y reutilizables.



## 9.2. SOAP vs REST

En lo que respecta a las metodologías de las API, existen principalmente dos enfoques: SOAP (Protocolo de Acceso a Objetos Simples) y REST (Representational State Transfer).

SOAP es un protocolo de comunicación que permite que las aplicaciones se comuniquen a través de la red. Utiliza XML para codificar sus mensajes y se basa en la especificación de servicios web estándar, que incluye WSDL (Web Services Description Language) para describir los servicios disponibles y UDDI (Universal Description, Discovery and Integration) para enumerar los servicios disponibles. SOAP puede operar sobre cualquier protocolo de transporte, como HTTP, SMTP o incluso JMS, y ofrece funciones avanzadas como la autenticación WS-Security.

Sin embargo, SOAP puede ser complejo y pesado para algunas aplicaciones, especialmente aquellas con limitaciones de ancho de banda o capacidad de procesamiento.

Por otro lado, REST es una arquitectura que utiliza HTTP y está basada en principios como stateless y la cacheabilidad. REST es más sencillo que SOAP y puede usar formatos de mensaje más pequeños y rápidos como JSON. REST es más adecuado para aplicaciones basadas en web y móviles que necesitan ser livianas y rápidas.

Sin embargo, REST puede no ser adecuado para aplicaciones que requieren una seguridad avanzada o transacciones a nivel empresarial, ya que no proporciona la misma gama de estándares y características que SOAP.

## 9.3. Windows Communication Foundation

Microsoft, ya desde 2002, propone un framework de desarrollo llamado .NET, un entorno donde se pueden crear y ejecutar aplicaciones en lenguajes como C# y Visual Basic entre otros. La propuesta de este entorno es la incorporación de bibliotecas con clases que definen todo tipo de objetos que se necesitan implementar en un desarrollo de software; como manejo de archivos, interfaz de usuario, bases de datos o redes entre una larga lista. .NET también propone su propia implementación de CLI (Common Language Infrastructure); CLR (Common Language Runtime o entorno en tiempo de ejecución de lenguaje común), el cual es el entorno de ejecución que se encarga de abstraer el lenguaje utilizado a la hora de compilar, de ahí que se puedan utilizar varios lenguajes, también se ocupa de administrar la memoria, los hilos y manejar excepciones entre otras cosas.

Antes de 2006, Microsoft proponía diferentes tecnologías independientes para la comunicación en aplicaciones distribuidas, tales como .NET Remoting, ASMX y Windows Communication Framework. Propuestas complejas de configurar por independiente, pero más aún de hacer que funcionasen juntas ya que cada tecnología tenía su propia configuración en términos propios y diferentes modelos de estructuración, lo que ocasionaba muy poca interoperabilidad.

Es con la llegada de .NET Framework 3.0, con la que se implanta una solución unificada; Windows Communication Foundation o WCF. Esta versión también traería unificaciones en otros aspectos, como Windows Presentation Foundation para las interfaces de usuario.

WCF es una solución de Microsoft para la comunicación entre aplicaciones en .NET, proporciona un marco para construir servicios y aplicaciones distribuidas, es altamente configurable y puede usar varios protocolos y estándares, incluyendo HTTP, TCP, Named Pipes, MSMQ y puede trabajar con SOAP y REST.

WCF tiene varias ventajas, como su flexibilidad y escalabilidad. Es posible crear servicios que puedan usar varios protocolos y que puedan ser accedidos por clientes construidos con una variedad de tecnologías. Sin embargo, la configuración de WCF puede ser compleja y la interoperabilidad con plataformas no .NET puede ser un infierno.

## 9.4. Web Services

Un servicio web es una tecnología que permite la comunicación e interacción entre sistemas a través de la red, proporcionando un mecanismo para la interoperabilidad de software. Un servicio web basado en SOAP a través de Windows Communication Foundation (WCF) es un tipo particular de API.

Al exponer un servicio web, WCF utiliza un contrato de servicio, que es esencialmente una interfaz definida por el programador que especifica qué operaciones están disponibles para ser consumidas por los clientes. Este contrato de servicio está descrito en XML a través del Lenguaje de Descripción de Servicios Web (WSDL). Este lenguaje es una forma estándar de describir cómo interactuar con un servicio web y detalla los métodos de entrada y salida, los tipos de datos y las rutas de acceso a los servicios.

Cuando un cliente consume un servicio web, utiliza este contrato de servicio para saber qué operaciones puede realizar y cómo realizarlas. El cliente envía un mensaje SOAP, que es simplemente un documento XML que sigue un formato específico, al servicio web. Este mensaje es transportado sobre el protocolo de transporte (como HTTP), y cuando el servicio web lo recibe, procesa el mensaje SOAP y realiza la operación correspondiente.

Aunque un servicio web puede proporcionar acceso a datos, no accede a los datos directamente. En lugar de eso, se sirve de dos patrones de diseño de software: el Objeto de Acceso a Datos (DAO) y el Objeto de Transferencia de Datos (DTO).

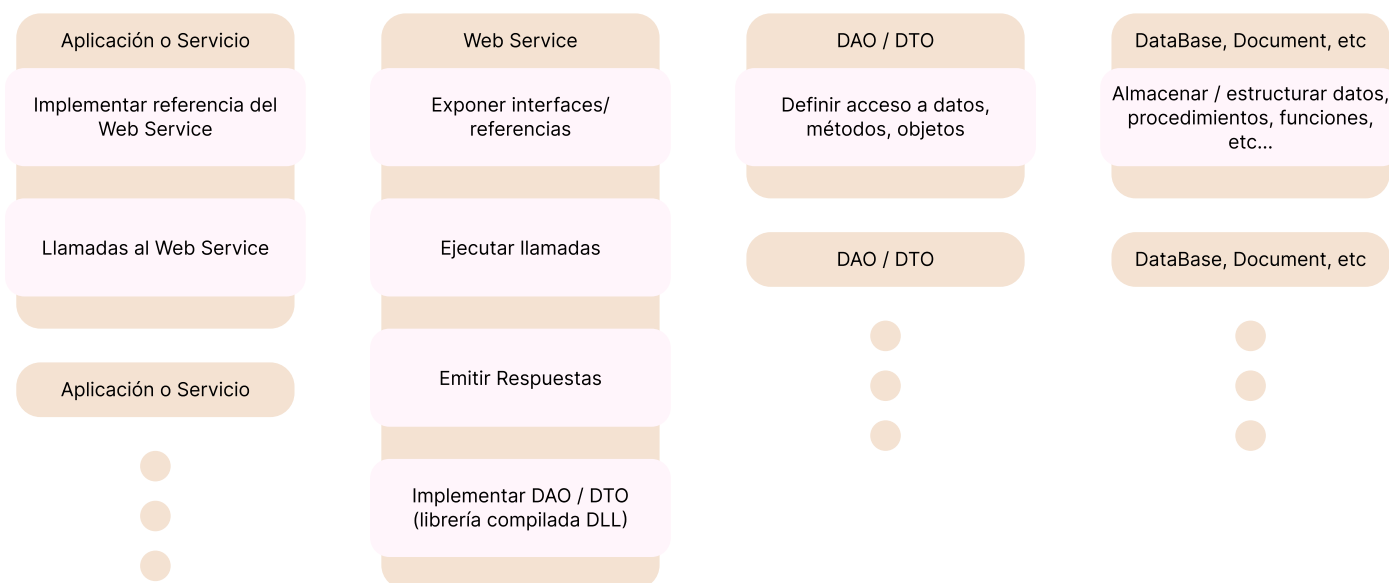
El DAO es un objeto que proporciona una interfaz abstracta a algún tipo de base de datos o mecanismo de persistencia. El DAO puede leer y escribir en la base de datos, pero el servicio web que lo utiliza no necesita saber los detalles de cómo se realiza esto. El DAO abstrae los detalles del acceso a datos y permite al servicio web centrarse en la lógica de negocio.

Por otro lado, el DTO es un objeto que se utiliza para agrupar varios datos que se pasan juntos. Un DTO puede ser utilizado por un DAO para pasar datos al servicio web. Una vez más, esto abstrae los detalles de cómo se almacenan los datos y permite al servicio web tratar los datos como un objeto coherente.

La abstracción proporcionada por el servicio web tiene beneficios significativos en términos de flexibilidad y modularidad. Al igual que diferentes aplicaciones y servicios pueden consumir el mismo servicio web sin tener que preocuparse por los detalles de su implementación, diferentes DAO pueden acceder a diferentes tipos de bases de datos y almacenar los datos de diferentes maneras.

Por ejemplo, un DAO podría acceder a una base de datos relacional y utilizar SQL para leer y escribir datos, mientras que otro DAO podría acceder a una base de datos NoSQL y utilizar una API de base de datos para acceder a los datos. Todo esto es transparente para el servicio web, que sólo ve el DAO y el DTO.

Esta abstracción significa que se puede cambiar la forma en que se accede y almacena los datos sin tener que cambiar el servicio web o las aplicaciones y servicios que lo consumen. Esto proporciona un alto grado de flexibilidad y permite a los desarrolladores seleccionar el mejor método de acceso a datos y almacenamiento de datos para sus necesidades específicas.



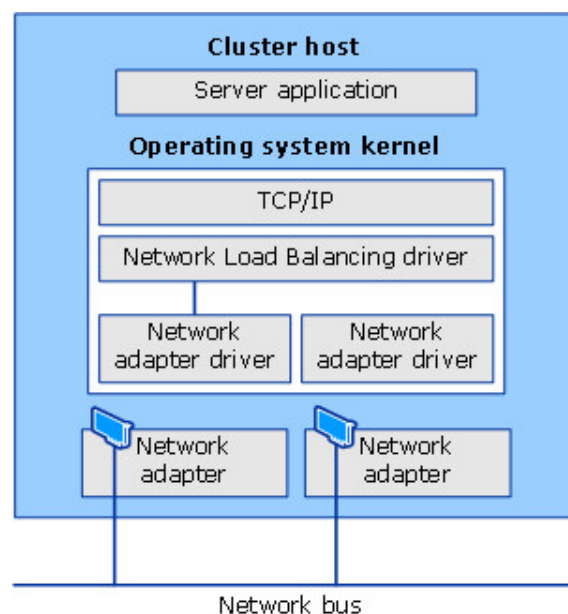
## 9.5. IIS Server y balanceo de carga

Internet Information Services (IIS) es un servidor web extensible y flexible creado por Microsoft para usar con Windows. Proporciona un conjunto sólido de características para cumplir con las necesidades más comunes de sitios web. Además, IIS es una parte crucial para muchas aplicaciones empresariales, desde servicios de correo electrónico y CRM hasta aplicaciones web.

La arquitectura de IIS es modular, lo que significa que las características se implementan a través de módulos que pueden ser añadidos o eliminados según sea necesario. Esto proporciona un alto grado de flexibilidad y personalización.

Para nuestro Web Service crearemos un sitio web en un equipo Windows Server con las características de IIS instaladas, en este, desplegaremos los archivos resultantes de la compilación del proyecto de .NET donde hemos desarrollado los diferentes módulos y recopilado los DAO como bibliotecas DLL. Esta tarea podemos realizarla mediante FTP o cualquier otro método para mover archivos, pero el propio Visual Studio tiene funciones para hacer este despliegue.

El balanceo de carga o NLB (Network Load Balancing) es una técnica con la que se distribuyen las solicitudes de los clientes entre diferentes servidores, operando de forma transparente para el cliente, es decir, el cliente solo ve un único servidor, con el NLB podemos decidir cual de los servidores responde a cada solicitud. Windows Server 2016 y superiores tienen la opción de añadir características específicas para llevar a cabo esta técnica. Los servidores que queramos implicar en este balanceo de carga, deben tener dos interfaces de red, una exterior y otra interna al cluster que formarán las máquinas.



En nuestro IIS tenemos que agregar características opcionales de NLB para que funcione nuestro planteamiento, además podemos definir el comportamiento de este balanceo, por ejemplo, usando round robin o eligiendo al servidor con más recursos libres, de cualquier modo lo importante es que configuremos que toda la comunicación con un mismo cliente la realice el mismo servidor, para lograr integridad en posibles tokens de sesión u otros parámetros que se deban mantener.

Al tener Windows Server todas estas características planteadas, no tenemos que preocuparnos de configuraciones como la traducción de puertos o NAT, estos detalles se configuran entre bambalinas al indicar nosotros las direcciones de los equipos.

## 9.6. Firewall

El firewall o cortafuegos es la barrera que delimita las comunicaciones de red de un sistema informático, digamos que es la aduana que controla los paquetes de red que recibe y envía nuestro equipo.

Con esta tecnología podemos filtrar los propios paquetes en base a reglas definidas, estas han de ser lo más restrictivas posibles, definiendo minuciosamente orígenes destinos, ya sea de direcciones como de puertos, contenido o flags de los paquetes, así como protocolos. También puede haber firewall a nivel de aplicación, pero con los filtrados de red podemos conseguir configuraciones muy robustas.

Para nuestro proyecto, por ejemplo, queremos redirigir tráfico HTTP a HTTPS. Así como solo escuchar peticiones provenientes de direcciones para las que se planean las comunicaciones, es decir, las peticiones a servicios destinados a usarse desde el almacén, no deberían atenderse si no provienen de la subred o conjunto de direcciones dedicadas a esa zona. Por supuesto hay que cerrar todos los puertos de servicios no relacionados con los servicios que ofrezca nuestro servidor.

## 10. Procedimientos

---

Aquí se dan a conocer un conjunto de pasos o una secuencia de actividades de forma lógica con la cual se conseguirán los objetivos pautados.

1. Identificar fuentes de datos y necesidades
2. Definir los DAO y DTO
3. Crear interfaces y servicios del Web Service
4. Desplegar el Web Service en IIS y configurar el balanceo de carga
5. Configurar firewall

## 11. Descripción de las Actividades

---

### 11.1. Identificar fuentes de datos y necesidades

La propuesta ideal de esta fase involucra a una persona responsable por cada servicio o aplicación que vaya a consumir nuestro Webservice, la responsable del proyecto tendrá que tener un par de sesiones para formar a cada responsable, estableciendo directrices y metodologías para plasmar las necesidades y la forma de operar de cada servicio o aplicación, así como instruirlos en cómo expresar dicha información en diagramas UML.

Una vez llevadas a cabo estas dos sesiones de un par de horas cada una y en días distintos para optimizar la adquisición de la información necesaria; habrá un plazo de una semana, donde cada responsable tendrá que documentar sus necesidades, tanto por escrito como en diagramas (se recomienda encarecidamente programas como Draw.io, donde los diagramas tienen un aspecto homogéneo y una amplia posibilidad de exportarlos y trabajar con ellos, por ejemplo XML, HTML, SVG, directamente a PDF y otro tipo de detalles que favorecen la interoperabilidad que se necesite para trabajar con ellos). Cumplido este plazo, la responsable del proyecto de Web Service evaluará la documentación y tendrá reuniones para aclarar posibles dudas y/o proponer mejoras, esta tarea tendrá una duración de entre una a tres semanas, según la cantidad de servicios y aplicaciones diferentes que se tengan que contemplar.

Para afianzar lo elaborado en esta fase se tendrá una reunión conjunta más, donde se hará una puesta en común de los resultados finales, tras ella otro lapso de una a dos semanas para pequeñas correcciones en pos de no repetir ningún tipo de método u objeto (siempre se pueden encontrar formas de compartir métodos o clases padres de las cuales partan las heredadas con pequeñas modificaciones).

En este punto podría dar comienzo la siguiente fase, pero si no existiese un sistema de usuarios coherente y con privilegios restrictivos sólo para la finalidad para la que se originan, esta tarea se debe realizar, con posibilidad de ser en paralelo a la segunda fase si se dispone del personal.

### 11.2. Definir DAO y DTO

Esta fase puede llegar a ser relativamente sencilla si se ha realizado correctamente la primera, ya que consiste en plasmar todas esas necesidades en cuanto a operaciones y objetos que se definían al inicio. Aquí ya entra en juego el uso de .NET y su IDE Visual Studio, donde se creará un proyecto que recoja cada componente separado en diferentes espacios de nombres, recogiendo sus clases y métodos. Por supuesto habrá código común, como por ejemplo los objetos de conexiones a bases de datos, podemos agruparlos todos en un espacio de nombres común y crear uno por cada fuente de datos diferente (vease diferentes bases de datos SQL, alguna base de datos no relacional, BBDD en la nube, etc), es muy probable que de una misma fuente de datos tengamos que crear diferentes objetos de conexión por cada perfil que acceda a ellos, siendo clases que partan del objeto de acceso a su fuente de datos; es decir, si tenemos tres usuarios distintos en un SQL Server, los tres heredarían la clase del objeto de conexión a ese servidor y añadirían lo que diferencia a cada acceso, como serían las credenciales. Esta parte obviamente requiere que el último paso de la fase anterior se complete, pero solo esta tarea, todo lo anterior puede realizarse sin problema.

Para desarrollar esta fase se estima una temporalización de dos semanas por cada cinco servicios o aplicaciones diferentes, contando con una semana base dedicada a los objetos de conexión. Para ir comprobando que el desarrollo funciona, se debería de ir compilando el código en librería DLL, incluirlas como dependencias en cualquier tipo de proyecto de .NET (Windows form puede ser la opción más rápida para poner en marcha), donde poder depurar los métodos y sus comportamientos. Podemos hacer uso de los componentes refiriéndonos a su espacio de nombres seguido de sus denominaciones, separado por un punto, por ejemplo "Inventario.ObtenerCantidadById". Habiendo terminado la depuración de un espacio de nombres al completo, podríamos comenzar con la tercera fase de ese componente, pero necesitaríamos tener dos personas involucradas en el desarrollo, si no, es mejor completar por completo esta fase antes de comenzar la siguiente.

### 11.3. Crear interfaces y servicios del Web Service

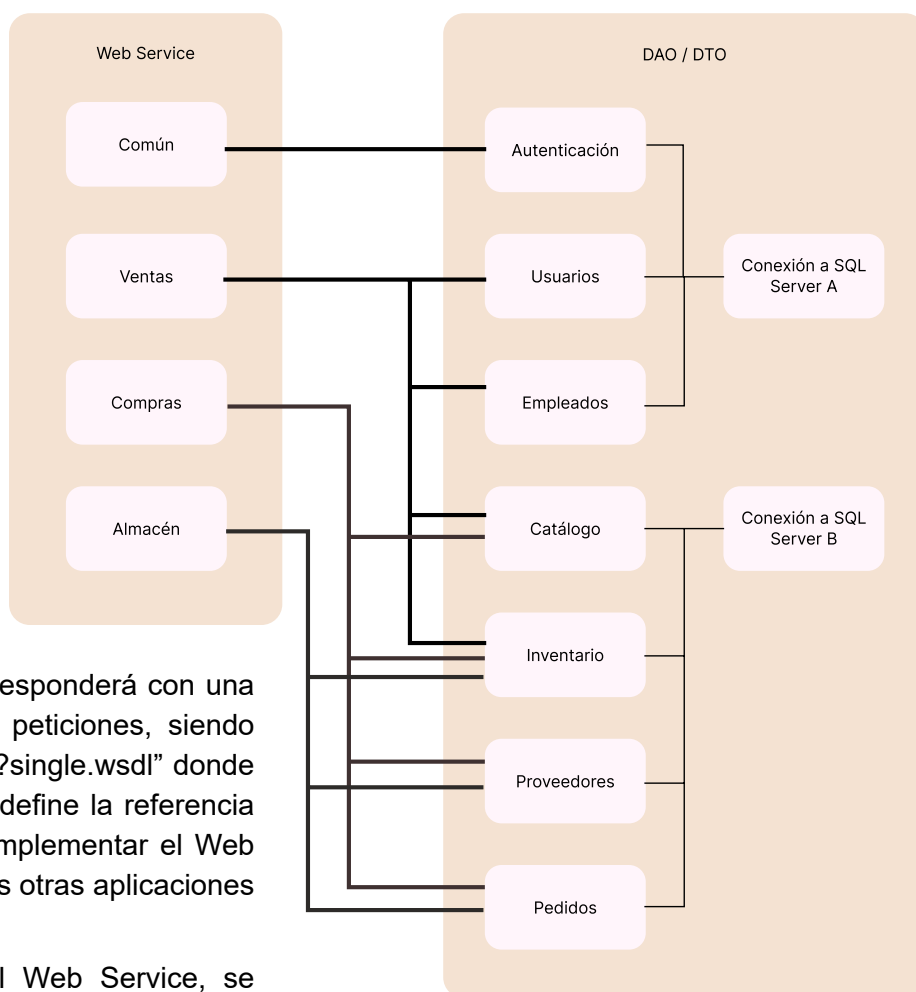
Esta fase necesita que la anterior se haya completado, o al menos que un componente completo se haya depurado y tengamos dos personas trabajando en este desarrollo.

Los Web Service, al fin de al cabo, son una página web que expone los métodos e interfaces, además de responder a las solicitudes de los clientes, por ello, el proyecto en el que son desarrollados será ASP.NET, el framework de .NET que está pensado para el desarrollo web, aunque en lugar de "páginas" creamos objetos "svc" o servicios, cada uno de estos archivos se corresponderá con una dirección URL donde escucha las peticiones, siendo esta misma dirección acabada en "?single.wsdl" donde se expone el documento XML que define la referencia de las interfaces necesarias para implementar el Web Service en los proyectos de nuestras otras aplicaciones o servicios.

Para depurar cada servicio del Web Service, se puede desplegar el servicio en la máquina local usando

el servidor IIS Express que implementa Visual Studio de serie, creando de nuevo una aplicación provisional que implemente las referencias (al incluir como referencia los documentos XML que expone el Web Service es como si incluyéramos una dependencia a una librería en nuestro proyecto) y consuma los servicios expuestos por el Web Service, pudiendo comprobar su correcto funcionamiento.

El tiempo consumido por esta fase se asemeja a la anterior, una semana base sumando dos semanas por cada cinco servicios contemplados.



#### 11.4. Desplegar Web Service en IIS y configurar balanceo de carga

La instalación y configuración de nuestro cluster Windows Server puede realizarse desde el inicio del proyecto, dejando documentado todo el proceso y decisiones tomadas. Las características necesarias para IIS y el sitio web también se pueden realizar independientemente al desarrollo del propio WS, dejando la información del despliegue lista para facilitar el paso del código compilado desde .NET a la ubicación del sitio web.

Al hacer el paso desde el equipo local de la desarrolladora al servidor o servidores, hay que tener en cuenta los cambios en las configuraciones que definen las direcciones a las que se realizan las peticiones del WS, así como los directorios, o cualquier aspecto que difiera de la depuración local al despliegue de producción. Esto solo se tiene que tener en cuenta al hacer la primera integración, ya que en adelante se pueden marcar dichos archivos de configuración para que no se actualicen en los siguientes despliegues a producción, algo similar a los que logramos con el archivo gitignore en los repositorios que usan Git.

Para depurar el correcto funcionamiento del despliegue de producción podemos servirnos de las mismas aplicaciones que fuesen usadas para depurar en local, pero apuntando ahora al servidor de producción, añadiendo configuraciones para comunicaciones HTTPS o certificados necesarios para que las peticiones sean aceptadas en producción.

Esta depuración del servicio en producción puede ir realizándose a la par que se depura con éxito en local, contando con que la infraestructura de servidores esté lista en ese momento.

#### 11.5. Configurar firewall

En el marco teórico ya exponíamos las necesidades que debe cubrir la configuración de nuestro firewall, tenemos que tener una buena documentación de la red de nuestra organización, de no ser así, habría que recopilar toda la información necesaria para poder definir las reglas de nuestro firewall, lo que podría alterar la temporalización de esta fase de forma dramática, según la complejidad y tamaño de nuestra red.

La traducción de puertos o NAT necesarios para el correcto funcionamiento del cluster de servidores deberían de configurarse en el firewall, pero como también hemos expuesto en el marco teórico, estas configuraciones las realiza Windows de forma transparente.

La comprobación y validación de nuestra configuración también puede ser una ardua tarea si no tenemos sistemas de monitorización de red en nuestra organización, esto sumado a la posible carencia de una documentación adecuada de nuestra red hacen que esta fase pueda durar desde menos de una semana a casi un mes si la envergadura de la infraestructura es grande o desorganizada.



## 12. Observaciones Finales

---

### 12.1. Posibles mejoras

Los microservicios representan una arquitectura que descompone las aplicaciones en componentes pequeños e independientes, cada uno de los cuales realiza una tarea o proceso específico. A diferencia de la arquitectura monolítica, en la que todos los componentes de la aplicación están entrelazados, los microservicios operan de forma independiente, lo que facilita su gestión, despliegue y escalabilidad.

La adopción de una arquitectura de microservicios puede proporcionar varios beneficios que podrían mejorar nuestra propuesta actual de Web Service de WCF. Uno de los principales beneficios es la escalabilidad. Como cada microservicio opera de forma independiente, se pueden escalar de forma individual para satisfacer las demandas del tráfico o las cargas de trabajo.

Además, usando microservicios también podemos aumentar la velocidad de desarrollo y despliegue de las aplicaciones. Dado que cada microservicio es independiente, podemos desarrollar, probar y desplegar cada aplicación por separado del resto del sistema, lo que significa que se pueden hacer cambios o añadir nuevas características sin afectar al resto de implementaciones.

Sin embargo, como con cualquier solución, los microservicios también tienen sus desventajas. Una de las más evidentes es la complejidad; la gestión de múltiples servicios independientes puede ser compleja, tanto desde el punto de vista del desarrollo como del despliegue y la monitorización. Además, los microservicios también pueden llevar a problemas de consistencia de datos, ya que cada servicio puede tener su propia base de datos.

Entre las soluciones profesionales más prominentes para la implementación de microservicios se encuentran Kubernetes y Docker.

- **Kubernetes:** Es una plataforma de código abierto para la automatización del despliegue, escalado y administración de aplicaciones en contenedores. Su principal ventaja radica en su extensibilidad y flexibilidad. Sin embargo, su curva de aprendizaje puede ser bastante poco amable.
- **Docker:** Es una plataforma para el desarrollo, envío y ejecución de aplicaciones en contenedores. Su mayor ventaja es su facilidad de uso y su amplio soporte en la comunidad de desarrollo. Sin embargo, Docker por sí mismo no maneja la orquestación de contenedores a gran escala, lo que significa que probablemente necesitarás una solución adicional como Kubernetes.

La tendencia en cuanto a los microservicios está fuertemente vinculada a las soluciones en la nube como pueden ofrecer Azure de Microsoft, Amazon Web Services o Google Cloud Platform, pero es un terreno para explorar en un proyecto puramente centrado en estas soluciones, por evitar confusiones, estas soluciones de cómputo en la nube disponen de implementaciones de Docker y Kubernetes, pero tienen sus ecosistemas planteados para que tu mismo montes conjuntos de microservicios que cubran tus necesidades, pero también que se monetizan de la forma más rentable para tu tipo de consumo, ya que podemos encontrar servicios que se pagan por cómputo, tiempo de actividad o una multitud de propuestas que define cada empresa.

## 13. Cronograma

---

### ANEXO 1



## 14. Webgrafía

---

### - Opciones de pago de Visual Studio

[Opciones de precios y compra | Visual Studio](#)

### - Introducción a WCF

[¿Qué es Windows Communication Foundation? - WCF | Microsoft Learn](#)

### - Comparativa entre WCF y alternativas

[Mi capa de servicios: ¿WCF o WEB API? - Kabel](#)

### - Artículo comparando SOAP y REST

[SOAP vs. REST: A Look at Two Different API Styles | Upwork](#)

### - API, definición Wikipedia

[API - Wikipedia, la enciclopedia libre](#)

### - Video ejemplo balanceo de carga

[\(233\) NLB - Balanceo de Carga en Windows 2016 Server - YouTube](#)

### - Ejemplo usando DAO y DTO

[Patrones de Diseño – Ejemplo DAO+DTO+Singleton,Búsqueda y filtro de datos -C#.SQL \(Cap 2\) – RJ Code Advance](#)

### - Video Data Access Object

[\(240\) El patrón DAO \(Data Access Object\). Manteniendo la persistencia de datos || UPV - YouTube](#)

### - NLB Windows Server

[Equilibrio de carga de red | Microsoft Learn](#)

### - Diagramas UML

[Diagrama UML: Qué es, cómo hacerlo y ejemplos | Miro](#)

## ANEXO 1

### DIAGRAMA DE GANTT

