Servicios de SharePoint 2013 - III: Programando un Servicio propio

Gustavo Velez

* + MVP de SharePoint.
  + Sitio web: <http://www.gavd.net>
  + Email: [gustavo@gavd.net](mailto:gustavo@gavd.net)

Juan Carlos González Martín, MVP de SharePoint

* + Co-fundador del Grupo de usuarios de SharePoint de España (SUGES, [www.suges.es](http://www.suges.es)) y del Grupo de usuarios de Cloud Computing de España (CLOUDES)
  + Twitter: @jcgm1978.
  + Blogs: <http://geeks.ms/blogs/ciin> & <http://jcgonzalezmartin.wordpress.com/>

Fabian Imaz, MVP de SharePoint

* + Director de la carrera SharePoint 2010 en Microsoft Virtual Academy, <http://www.mslatam.com/latam/technet/mva2/Home.aspx>, y cuenta con un sitio en CodePlex con varios desarrollos (<http://siderys.codeplex.com>).
  + Twitter: @fabianimaz
  + Blog: <http://blog.siderys.com>

## Introducción

Esta serie de artículos discute la funcionalidad de Servicios en Microsoft SharePoint 2013. En el tercer número se habla de la capacidad de crear un servicio personalizado, instalarlo en SharePoint y consumir el mismo desde las diferentes Aplicaciones Web.

La arquitectura de servicios de SharePoint 2013 permite construir nuevas aplicaciones de servicio con el fin de incorporar nuevas funcionalidades a la plataforma. Estas nuevas funcionalidades pueden ser muy diversas logrando así que SharePoint se transforme en una plataforma base dentro de nuestra organización y brindándole a los usuarios un única herramienta para trabajar. Muchas organizaciones cuenta con sistemas de gestión, facturación, de recursos humanos, de manejo de stock, monitoreo y mucho más, donde los usuarios para poder realizar sus tareas día tras días necesitan ingresar a todos estos sistemas y muchas veces realizar la consolidación de datos requiere la implementación de otras herramientas o muchas horas de trabajo. La realidad de estas organizaciones sería diferente si en vez de implementar estas distintas plataformas, que incluso muchas veces están en tecnologías diferentes, decidieran implementar servicios y aplicaciones de servicios en SharePoint 2013.

## Elementos que componen una aplicación de servicio

Cada aplicación de servicio se compone de una serie de bloques necesarios para ejecutar los mismos dentro de la granja de SharePoint 2013. A continuación se detallan los elementos mínimos que son necesarios en una aplicación de servicio personalizada:

**Servicio:** Este elemento es el servicio en sí mismo, es el componente que estará ejecutándose una vez instalado, pudiendo ser un servicio de Windows o un servicio de WCF. Puede estar hospedado en uno o varios servidores de la granja. Si bien no es necesario la utilización de WCF como FrameWork de comunicación, es ideal para poder implementar una seguridad robusta dada todas las ventajas que esta tecnología provee.

**Instancia del Servicio:** Representa una instancia del servicio corriendo en uno de los servidores de la granja. Podemos tener múltiples instancias corriendo en varios servidores de la granja de SharePoint 2013 y gracias a la arquitectura provista por las aplicaciones de servicio, las instancias se ejecutan en entornos escalables y balanceados que permiten su correcto funcionamiento.

**Aplicación de Servicio:** Representa lógicamente al servicio dentro de la granja de SharePoint, encapsulando el servicio físico que hemos instalado en la granja sin importar en cuantos servidores hemos activado el mismo, todos los componentes de SharePoint utilizarán está abstracción cuando deban interactuar con el servicio.

**Base de datos del Servicio:** En algunos servicios se necesita tener una estructura de almacenamiento como una base de datos para su correcto funcionamiento. Esta base de datos será creada o al momento de activar el servicio en la granja de SharePoint o bien al crear la aplicación de servicio dentro de SharePoint.

**Proxy del Servicio de Aplicación:** Al crear una aplicación de servicio en SharePoint se crea también un Proxy, una estructura de comunicación que es expuesta por la granja a través de un servicio de WCF para que todos los componentes web (Elementos webs, Flujos, Eventos, aplicaciones personalizadas, etc) puedan consumir las operaciones del servicios desde la capa de presentación o servidor web.

Cada vez que se instala SharePoint 2013 este instala y configura una serie servicios de aplicación que serán utilizados para poder consumir correctamente todos los servicios de que se configuren y además provee una serié páginas para poder administrar los mismos:

**Descubrimiento de Aplicaciones y Balanceador de carga**: Esta aplicación de servicio es utilizada para hacer un correcto descubrimiento de todas las aplicaciones de servicio configuradas en la granja de SharePoint y manejar todas las peticiones que se realizan entre las distintas instancias instaladas.

**Identificadores de Seguridad (Security Token):** Este servicio provee todo el mecanismo de autenticación entre las diferentes aplicaciones y capas de la granja de SharePoint.

**Manejo de seguridad y acceso:** Los servicios de aplicación cuentan con niveles de seguridad que permiten establecer quienes serán los administradores de los mismos, por defecto los usuarios que son Administradores de la granja pueden acceder a todas las aplicaciones de servicio creadas, pero en un ambiente mixto se puede tener la administración de cada una de las aplicaciones de servicio descentralizada y de esta forma, asignar a un usuario o grupo de usuarios por ejemplo la administración de una aplicación de servicio en particular.

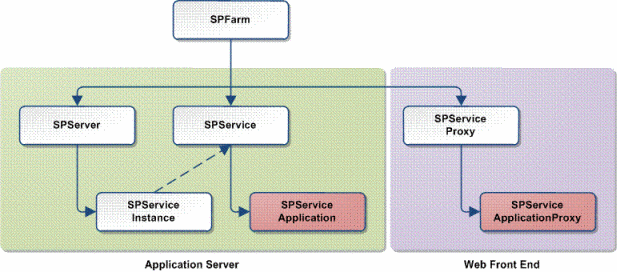
**Páginas de administración:** Todas las aplicaciones de servicio cuentan con una serié de páginas necesarias para poder administrar las mismas, cambiarlas o editar sus propiedades. Dependiendo de cada aplicación de servicio estás páginas podrán variar según los parámetros y opciones de cada aplicación.

## El modelo de objetos

En SharePoint 2013 contamos con un Framework de clases para que los desarrolladores puedan construir sus propios servicios de aplicaciones e instalarlos. Todas las aplicaciones de servicios provistas por SharePoint fueron creadas basadas en este Framework el cual podemos ver en la figura 1.

* 1. Figura 1. - Relación entre las clases del Framework del servicio de aplicación

Fuente [**MSDN**](http://msdn.microsoft.com/en-us/library/cc768564.aspx)

* 1. 

A continuación se describen brevemente cada una de las clases involucradas en el Framework provisto para la construcción de servicios de aplicaciones.

[SPFarm:](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.spfarm.aspx) Esta clase representa la granja de SharePoint 2010 dentro del modelo de objetos y es la clase más alta en la jerarquía de clases provista, brindando acceso a la topología de servidores dentro de la granja o los servicios configurados en la misma por ejemplo.

[SPServer:](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.spserver.aspx) A través de esta clase se obtiene acceso al servidor físico dentro de la granja pudiendo habilitar por ejemplo servicios que deberán correr en el mismo.

[SPService:](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.spservice.aspx) A través de esta clase se puede tener acceso al servicio instalado en la granja de SharePoint. Para acceder a todos los servicios disponibles debemos obtener la colección [SPServiceCollection](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.spservicecollection.aspx) de servicios a través de la propiedad [Services](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.spfarm.services.aspxhttp:/msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.spfarm.services.aspx) del objeto [SPFarm](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.spfarm.aspx).

[SPServiceInstance:](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.spserviceinstance.aspx) Esta clase representa la instancia del servicio corriendo en un servidor de la granja. Puesto que se puede tener configurado varias instancias del mismo servicio en distintos servidores de la granja a través de la propiedad [Instances](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.spservice.instances.aspx) de la clase [SPservice](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.spservice.aspx) podemos tener todas las instancias de dicho servicio que fueron configuradas.

[SPServiceApplication:](http://msdn.microsoft.com/es-ar/library/microsoft.sharepoint.administration.spserviceapplication.aspx) Esta clase representa a la aplicación de servicio que se puede crear en SharePoint 2013 para exponer las funcionalidades del servicio que tenemos configurado en la granja. Cada aplicación de servicio creada corre de forma aislada de otra aplicación de servicio del mismo tipo, manteniendo sus propias bases de datos y su configuración de seguridad, permitiendo así tener por ejemplo varias aplicaciones de servicios de búsqueda donde los elementos que se están rastreando no serán compartidos entre sí.

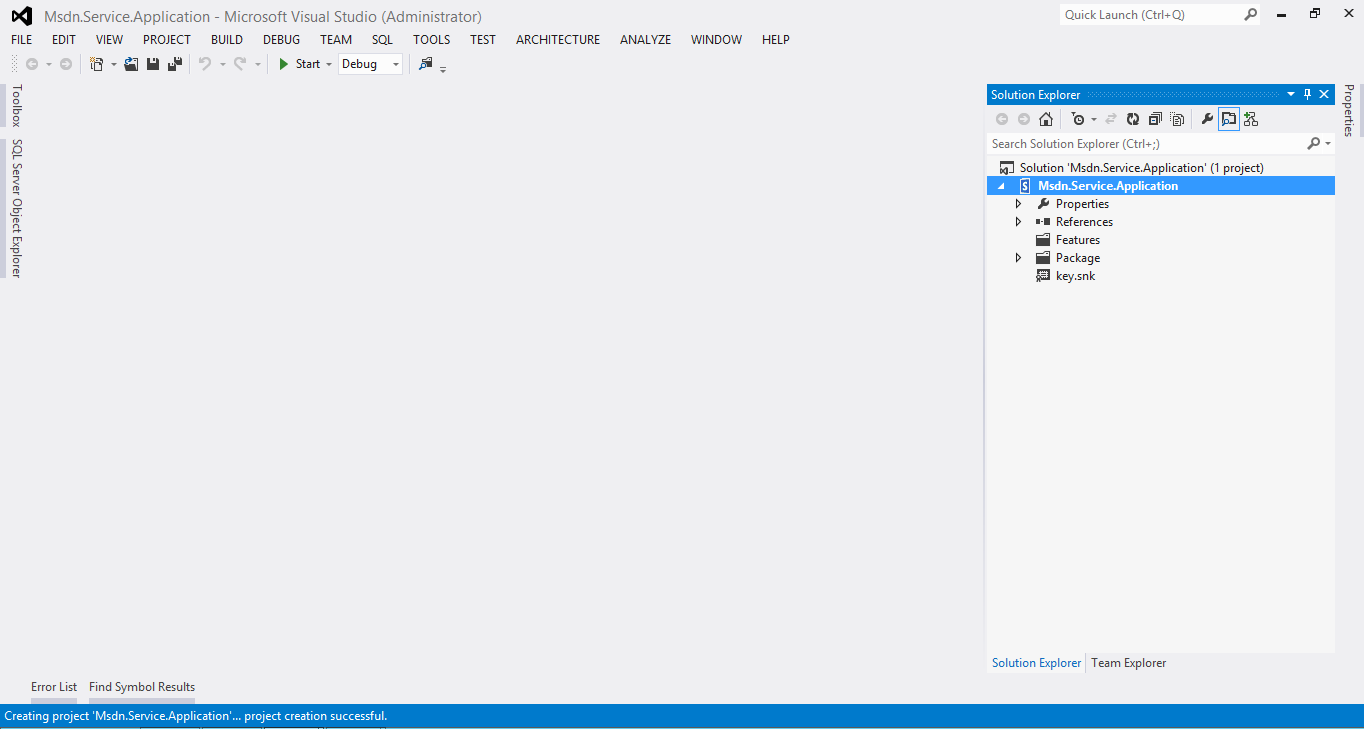
[SPServiceProxy:](http://msdn.microsoft.com/es-ar/library/microsoft.sharepoint.administration.spserviceproxy.aspx) Esta es la clase base cliente que se utiliza para exponer un servicio de aplicación como hacia las aplicaciones web de SharePoint que tengan asociado dicho servicio.

[SPServiceApplicationProxy:](http://msdn.microsoft.com/es-ar/library/microsoft.sharepoint.administration.spserviceapplicationproxy.aspx) Esta clase expone todas las interfaces públicas que provee un servicio de aplicación para que cualquier aplicación web (Front End) de SharePoint pueda consumir. Cualquier elemento web, evento, workflow o artefacto que es hospedado en una aplicación web de SharePoint y necesita consumir un servicio de aplicación configurado en SharePoint deberá consumirlo a través de la clase cliente creada automáticamente cuando se creó el servicio de aplicación.

## Programando un Servicio Propio

En este caso se va a implementar un servicio dentro de SharePoint 2013 que brinde la posibilidad de extender la plataforma de manera que los usuarios puedan manejar el sistema de gestión de la organización directamente desde SharePoint 2013. Este servicio permitirá a los usuarios manejar “las órdenes de ventas” del sistema de gestión a través de SharePoint. Antes de comenzar, instale en su servidor de base de datos (BD) la BD “AdventureWorks2012” que puede descargarse desde sitio de [CodePlex](http://msftdbprodsamples.codeplex.com/releases/view/55330).

El primer paso para crear una nueva aplicación de servicio pasa por crear un proyecto en blanco de SharePoint 2013 en nuestro Visual Studio 2013.

* 1. Figura 2. - Proyecto en blanco de SharePoint 2013
  2. 

Una decisión importante que se debe tomar es el lugar donde se implementará la lógica del servicio que se está construyendo. En el ejemplo, se va a construir un servicio que exponga operaciones de la lógica de negocios del sistema de gestión organizacional a través de WCF lo que implica definir los contratos y tipos de entidades que estás operaciones. Podemos desarrollar todas nuestras clases en un mismo ensamblado que será el servicio que se está implementando, pero esto implica que cada vez que se realice un cambio en la lógica del negocio es necesario recompilar completamente el servicio e implementarlo de nuevo, por lo que es recomendable implementar la lógica de negocios en ensamblados diferentes. Por lo tanto, se añadirán 5 proyectos más en Visual Studio 2012 de la siguiente manera:

Proyecto de Biblioteca de Clases para definir todas las entidades que vamos a estar usando.

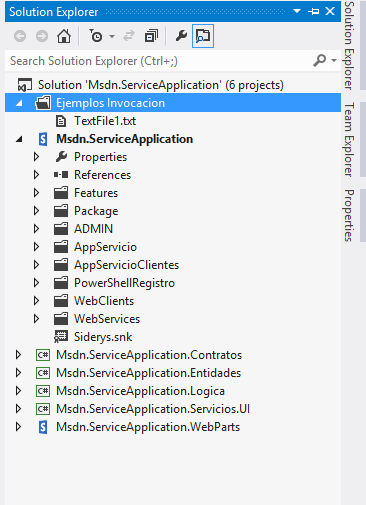
Proyecto de Biblioteca de Clases para definir los contratos

Proyecto de Biblioteca de Clases para implementar las operaciones de negocio.

Proyecto de Biblioteca de Clases donde implementaremos los servicios que se le brindaran a la capa de presentación, UI, para que encapsule las llamadas al servicio en SharePoint.

Proyecto de SharePoint donde crearemos todos los componentes web que serán utilizados por los usuarios finales en un portal de SharePoint 2010.

Una vez definidos todos los proyectos, nuestra solución debería quedar como se muestra en la siguiente imagen.

* 1. Figura 3. - Proyectos para la implementación de la aplicación de servicio
  2. 

Antes de ponernos a implementar el servicio y las operaciones que estará exponiendo, para ser utilizado desde SharePoint se deben crear algunas carpetas necesarias dentro del proyecto del servicio y mapearlas con carpetas del directorio de instalación de SharePoint 2013. A continuación se presenta la lista de carpetas que se necesitan:

**Admin** – Contiene todas las páginas necesarias para manejar la aplicación de servicio desde la administración central de SharePoint, creación, propiedades y manejo del servicio. La ruta física de esta carpeta se encuentra en {SharePointRoot}\Template\ADMIN.

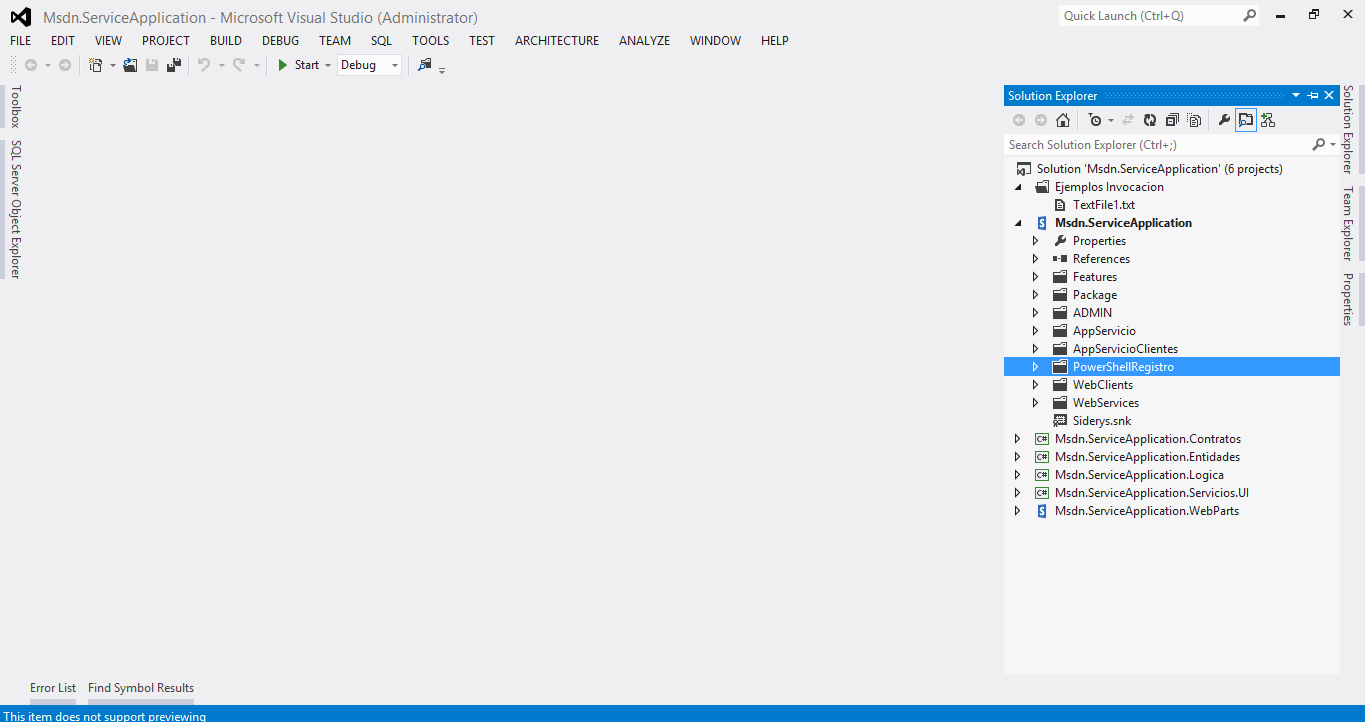
**PowerShellRegistro** - Contiene los archivos de configuración para registrar los comandos PowerShell necesarios para poder crear la aplicación de servicio por línea de comandos. La ruta física de esta carpeta se encuentra en {SharePointRoot}\CONFIG\POWERSHELL\Registration.

**SQL** – En esta carpeta se ubican los archivos necesarios para crear la base de datos de la aplicación de servicio en el caso que así lo necesitara. Dependiendo del tipo de servicio que se esté desarrollando podrá tener una base de datos propia donde almacenar la información que este deba persistir. En este ejemplo no es necesario que el servicio disponga de una base de datos ya que se usará la base datos externa de nuestro sistema de gestión. La ruta física de esta carpeta se encuentra en {SharePointRoot}\Template\SQL.

**WebServices** – En esta carpeta se almacenara los archivos necesarios para exponer las operaciones de los servicios a través de WCF. Básicamente para el servicio de ejemplo que se va a construir, esta carpeta contiene el archivo .svc y el web.config del servicio con la definiciones de cómo serán expuestas las operaciones. La ruta física de esta carpeta se encuentra en {SharePointRoot}\WebServices.

**WebClients** – En esta carpeta se almacenan los archivos necesarios para que los clientes puedan consumir los servicios expuestos por WCF. Contiene un archivo de configuración que define como y donde se expusieron las operaciones que el servicio expone. La ruta física de esta carpeta se encuentra en {SharePointRoot}\WebClients.

En la Figura 4 se ver como quedo armado toda la solución en Visual Studio 2012 para poder empezar a codificar la aplicación de servicio.

* 1. Figura 4. - Capetas mapeadas al directorio de instalación de SharePoint
  2. 

Una vez que todos los componentes necesarios fueron creados, vamos a comenzar a implementar todas las clases necesarias para poder crear la aplicación de servicio. La primera clase que se va a implementar es la clase del servicio, que en este caso particular clase heredar de la clase abstracta [SPIisWebService](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.spiiswebservice.aspx) que hereda a su vez de SPService (**Nota:** Recuerde que esta clase en el diagrama de clases de la Figura 1, es la clase base del FrameWork provisto por SharePoint). También se implementara la interfaz [IServiceAdministration](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.iserviceadministration.aspx) la cual obliga a implementar los métodos administrativos para la creación de los componentes necesarios para crear el servicio de aplicación.

* 1. Listado 1. - Clase ServicioGestion
  2. namespace Msdn.ServiceApplication
  3. {
  4. [Guid("A14F914A-7701-4fa8-83B6-975D37AEC410")]
  5. public class ServicioGestion : SPIisWebService, IServiceAdministration
  6. {
  7. public ServicioGestion()
  8. : base()
  9. { }
  10. public ServicioGestion(SPFarm pGranja)
  11. : base(pGranja)
  12. { this.Name = "MSDN Servicio de Gestion"; }
  13. public SPServiceApplication CreateApplication(string name, Type serviceApplicationType, SPServiceProvisioningContext provisioningContext)
  14. {
  15. if (provisioningContext == null)
  16. throw new ArgumentNullException("provisioningContext");
  17. if (serviceApplicationType != typeof(ServicioGestionAplicacion))
  18. throw new NotSupportedException("Tipo de servicio de aplicacion invalido, se esperaba AplicacionServicioGestion");
  19. ServicioGestionAplicacion lapplication = (ServicioGestionAplicacion)this.Farm.GetObject(name, this.Id, serviceApplicationType);
  20. if (lapplication == null)
  21. {
  22. lapplication = ServicioGestionAplicacion.Create(name, this, provisioningContext.IisWebServiceApplicationPool);
  23. }
  24. return lapplication;
  25. }
  26. public SPServiceApplicationProxy CreateProxy(string pNombre, SPServiceApplication pAplicacionServicio, SPServiceProvisioningContext pContexto)
  27. {
  28. if (pAplicacionServicio == null)
  29. throw new ArgumentNullException("serviceApplication");
  30. if (pAplicacionServicio.GetType() != typeof(ServicioGestionAplicacion))
  31. throw new NotSupportedException("Tipo de aplicación de servicio invalido");
  32. ServicioGestionProxy lServicioProxy = (ServicioGestionProxy)base.Farm.GetObject(string.Empty, base.Farm.Id, typeof(ServicioGestionProxy));
  33. if (lServicioProxy == null)
  34. {
  35. lServicioProxy = new ServicioGestionProxy(base.Farm);
  36. lServicioProxy.Update(true);
  37. }
  38. ServicioGestionAplicacionProxy lAplicacionProxy = lServicioProxy.ApplicationProxies.GetValue<ServicioGestionAplicacionProxy>(pNombre);
  39. if (lAplicacionProxy == null)
  40. {
  41. lAplicacionProxy = new ServicioGestionAplicacionProxy(pNombre, lServicioProxy, ((ServicioGestionAplicacion)pAplicacionServicio).Uri);
  42. }
  43. lAplicacionProxy.Update(true);
  44. return lAplicacionProxy;
  45. }
  46. public SPPersistedTypeDescription GetApplicationTypeDescription(Type serviceApplicationType)
  47. {
  48. if (serviceApplicationType != typeof(ServicioGestionAplicacion))
  49. throw new NotSupportedException("Tipo de aplicación de servicio invalido, se esperaba AplicacionServicioGestion");
  50. return new SPPersistedTypeDescription("Servicio de aplicacion de gestion", "Servicio para el manejo del sistema de gestion");
  51. }
  52. public Type[] GetApplicationTypes()
  53. {
  54. return (new Type[] { typeof(ServicioGestionAplicacion) });
  55. }
  56. public override SPAdministrationLink GetCreateApplicationLink(Type serviceApplicationType)
  57. {
  58. return new SPAdministrationLink("/\_admin/ServicioGestion/ServicioGestionCreateApp.aspx");
  59. }
  60. SPCreateApplicationOptions GetCreateApplicationOptions(Type serviceApplicationType)
  61. {
  62. return SPCreateApplicationOptions.None;
  63. }
  64. }
  65. }

El próximo paso consiste en crear la clase que representa la instancia del servicio, esta clase deberá heredar de la clase abstracta [SPIisWebServiceInstance](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.spiiswebserviceinstance.aspx), como se puede ver en la Listado 2.

* 1. Listado 2. - Clase InstanciaServicioGestion
  2. namespace Msdn.ServiceApplication
  3. {
  4. [Guid("14BCA048-41C0-4d64-AE05-52D8656404F2")]
  5. public class ServicioGestionInstancia : SPIisWebServiceInstance
  6. {
  7. public ServicioGestionInstancia()
  8. : base()
  9. { }
  10. internal ServicioGestionInstancia(SPServer pServidor, ServicioGestion pServicio)
  11. : base(pServidor, pServicio)
  12. {
  13. }
  14. internal ServicioGestionInstancia(string pNombre, SPServer pServidor, ServicioGestion pServicio)
  15. : base(pServidor, pServicio)
  16. {
  17. this.Name = pNombre;
  18. }
  19. public override string DisplayName
  20. {
  21. get { return "MSDN Servicio de Gestion"; }
  22. }
  23. public override string TypeName
  24. {
  25. get { return "MSDN Servicio de Gestion"; }
  26. }
  27. }
  28. }

A continuación hay que crear la clase que representara la aplicación de servicio en SharePoint 2013 la cual deberá heredar de la clase abstracta [SPIisWebServiceApplication](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.spiiswebserviceapplication.aspx). Esta clase expone las operaciones que serán consumidas desde el cliente o incluso desde algún otro servicio desarrollado y hospedado en SharePoint 2013.

* 1. Listado 3. - Clase ServicioGestionAplicacion
  2. namespace Msdn.ServiceApplication
  3. {
  4. [IisWebServiceApplicationBackupBehavior, Guid("E1B9DA13-DB64-4e4f-B296-D51FA01711AE")]
  5. public class ServicioGestionAplicacion : SPIisWebServiceApplication, ICliente, IOrdenesDeVenta, IProducto
  6. {
  7. [Persisted]
  8. private string mCadenaConexion = "Data Source=yoda;Initial Catalog=AdventureWorks2008R2;Integrated Security=True";
  9. internal string CadenaConexion
  10. {
  11. get { return mCadenaConexion; }
  12. set { mCadenaConexion = value; }
  13. }
  14. public ServicioGestionAplicacion()
  15. { }
  16. internal ServicioGestionAplicacion(string pNombre, ServicioGestion pServicio, SPIisWebServiceApplicationPool pApplicationPool)
  17. : base(pNombre, pServicio, pApplicationPool)
  18. {
  19. }
  20. public static ServicioGestionAplicacion Create(string pNombre, ServicioGestion pServicio, SPIisWebServiceApplicationPool pPoolDeAplicaciones)
  21. {
  22. if (pNombre == null)
  23. throw new ArgumentNullException("Nombbre");
  24. if (pServicio == null)
  25. throw new ArgumentNullException("Servicio");
  26. if (pPoolDeAplicaciones == null)
  27. throw new ArgumentNullException("Pool de aplicaciones");
  28. ServicioGestionAplicacion lAplicacionServicioGestion = new ServicioGestionAplicacion(pNombre, pServicio, pPoolDeAplicaciones);
  29. lAplicacionServicioGestion.Update();
  30. lAplicacionServicioGestion.AddServiceEndpoint("http", SPIisWebServiceBindingType.Http);
  31. lAplicacionServicioGestion.AddServiceEndpoint("https", SPIisWebServiceBindingType.Https, "secure");
  32. lAplicacionServicioGestion.Update();
  33. //creo la instancia para el sistema de gestion.
  34. return lAplicacionServicioGestion;
  35. }
  36. protected override string DefaultEndpointName
  37. {
  38. get { return ("http"); }
  39. }
  40. public override string TypeName
  41. {
  42. get { return ("Servicio de Aplicacion de Gestion"); }
  43. }
  44. protected override string InstallPath
  45. {
  46. get { return SPUtility.GetGenericSetupPath(@"WebServices\ServicioGestion"); }
  47. }
  48. protected override string VirtualPath
  49. {
  50. get
  51. {
  52. return "SevicioGestion.svc";
  53. }
  54. }
  55. public override Guid ApplicationClassId
  56. {
  57. get { return (new Guid("F624D941-757A-4619-B635-0C9D8F8BD6A5")); }
  58. }
  59. public override Version ApplicationVersion
  60. {
  61. get { return (new Version("1.0.0.0")); }
  62. }
  63. public override void Provision()
  64. {
  65. base.Status = SPObjectStatus.Provisioning;
  66. this.Update();
  67. //this.Provision();
  68. this.Status = SPObjectStatus.Online;
  69. this.Update();
  70. }
  71. public override void Unprovision(bool deleteData)
  72. {
  73. base.Status = SPObjectStatus.Unprovisioning;
  74. this.Update();
  75. base.Unprovision(deleteData);
  76. base.Status = SPObjectStatus.Disabled;
  77. this.Update();
  78. }
  79. public override SPAdministrationLink ManageLink
  80. {
  81. get
  82. {
  83. return new SPAdministrationLink("/\_admin/ServicioGestion/ServicioGestionManejarApp.aspx?id=" + this.Id.ToString());
  84. }
  85. }
  86. public override SPAdministrationLink PropertiesLink
  87. {
  88. get
  89. {
  90. return new SPAdministrationLink("/\_admin/ServicioGestion/ServicioGestionPropiedadesApp.aspx?id=" + this.Id.ToString());
  91. }
  92. }
  93. protected override SPNamedCentralAdministrationRights[] AdministrationAccessRights
  94. {
  95. get
  96. {
  97. return base.AdministrationAccessRights;
  98. }
  99. }
  100. protected override SPNamedIisWebServiceApplicationRights[] AccessRights
  101. {
  102. get
  103. {
  104. return new SPNamedIisWebServiceApplicationRights[]
  105. {
  106. SPNamedIisWebServiceApplicationRights.FullControl,
  107. new SPNamedIisWebServiceApplicationRights("Request", ServicioGestionDerechoAcceso.Request),
  108. SPNamedIisWebServiceApplicationRights.Read,
  109. };
  110. }
  111. }
  112. #region Operaciones Gestion
  114. public List<Cliente> ObtenerTodosLosClientes()
  115. {
  116. LogicaClientes lLogica = new LogicaClientes(mCadenaConexion);
  117. return lLogica.ObtenerTodosLosClientes();
  118. }
  119. public Cliente ObtenerClientePorId(int pIdCliente)
  120. {
  121. LogicaClientes lLogica = new LogicaClientes(mCadenaConexion);
  122. return lLogica.ObtenerClientePorId(pIdCliente);
  123. }
  124. public List<OrdenDeVenta> ObtenerTodasLasOrdenesDeVenta()
  125. {
  126. Logica.LogicaOrdenesDeVenta lLogicaOrdenesDeVenta = new Logica.LogicaOrdenesDeVenta(mCadenaConexion);
  127. return lLogicaOrdenesDeVenta.ObtenerTodasLasOrdenesDeVenta();
  128. }
  129. public List<OrdenDeVenta> ObtenerOrdenesDeVentaPorIdCliente(int pIdCliente)
  130. {
  131. Logica.LogicaOrdenesDeVenta lLogicaOrdenesDeVenta = new Logica.LogicaOrdenesDeVenta(mCadenaConexion);
  132. return lLogicaOrdenesDeVenta.ObtenerOrdenesDeVentaPorIdCliente(pIdCliente);
  133. }
  134. public void IngresarOrdenDeVenta(OrdenDeVenta pOrdenDeVenta)
  135. {
  137. }
  138. public void ElimindarOrdenDeVenta(OrdenDeVenta pOrdenDeVenta)
  139. {
  141. }
  142. public List<Producto> ObtenerTodosLosProductos()
  143. {
  144. LogicaProductos lLogica = new LogicaProductos(mCadenaConexion);
  145. return lLogica.ObtenerTodosLosProductos();
  146. }
  147. #endregion
  148. }
  149. }

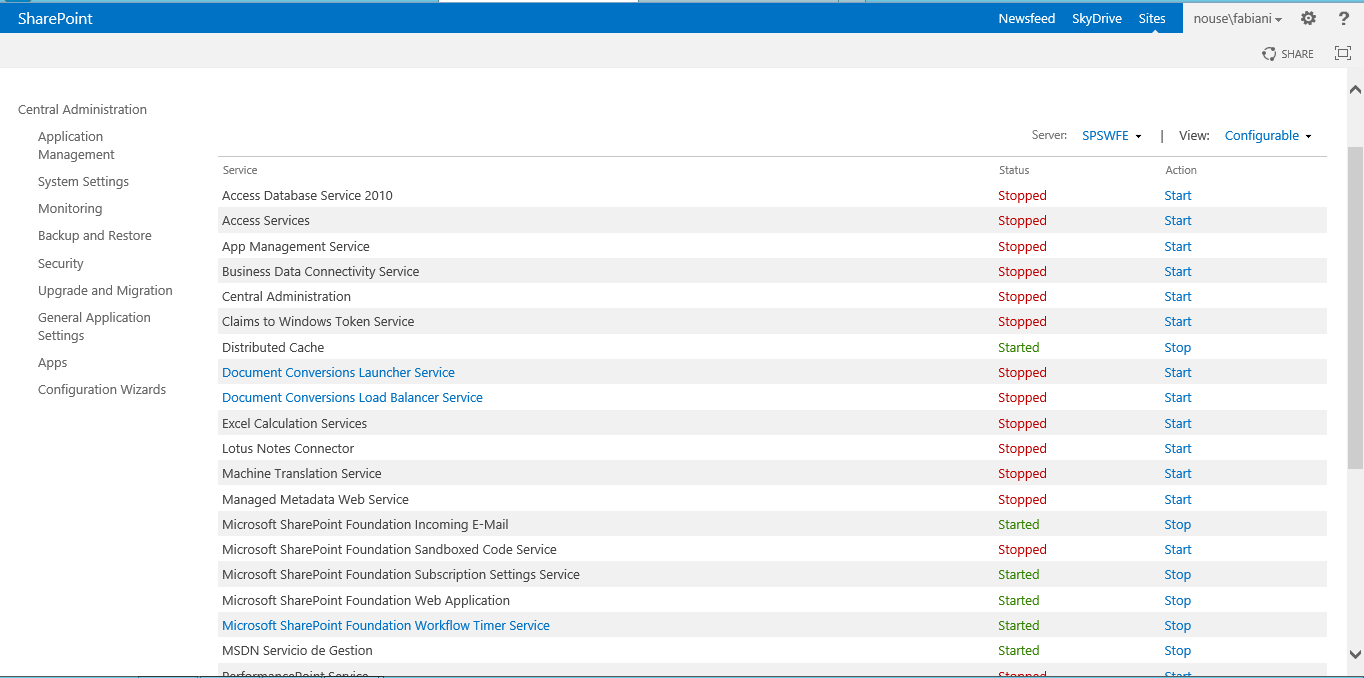
Una vez creada las clases que representan los componentes del lado del servidor, deberemos crear todos los componentes que expondrán los servicios a través de un proxy. Como se está creando un servicio que expone operaciones de negocios implementadas en WCF, sé tiene que exponer primero el servicio a través del IIS, para lo cual se va a crear una clase de servicio .SVC el cual expondrá todas las operaciones. Pero estas operaciones estarán implementadas en la clase “ServicioGestionAplicacion” por lo cual deberemos declarar en el archivo que define el servicio que clase implementa las operaciones y cuál será la clase de creación del mismo, en la sección 4 se puede ver la definición del mismo. Además se deberá realizar las configuraciones de WCF para establecer como estará expuesto el servicio en nuestro servidor y la configuración para los clientes de como deberán consumir los mismos.

* 1. Listado 4. - SevicioGestion.svc
  2. <%@ServiceHost language=c# Debug="true"
  3. Service="Msdn.ServiceApplication.ServicioGestionAplicacion, Msdn.ServiceApplication, Version=1.0.0.0, Culture=neutral, PublicKeyToken=711eed342842acee"
  4. Factory="Msdn.ServiceApplication.ServicioGestionApplicationHostFactory, Msdn.ServiceApplication, Version=1.0.0.0, Culture=neutral, PublicKeyToken=711eed342842acee"
  5. %>

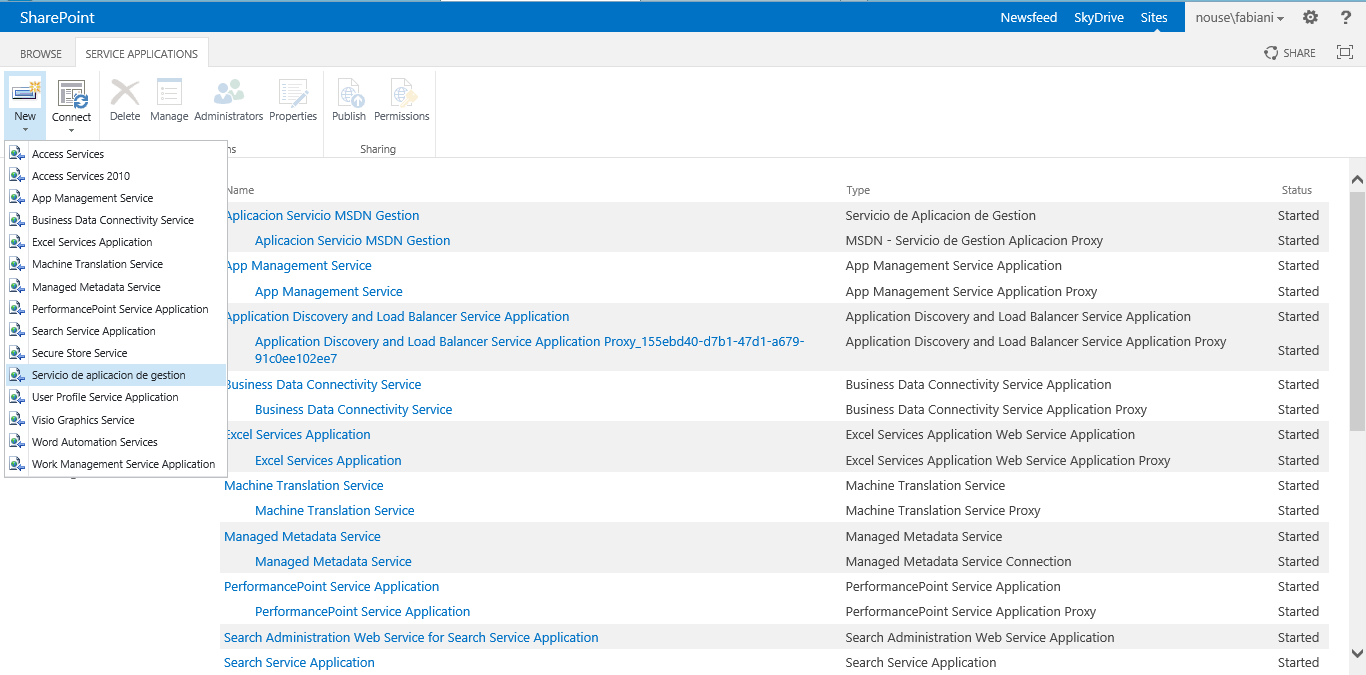
Lo próximo que se debe crear son las clases del tipo proxy, clases que expondrán el servicio a los clientes web y que estos podrán consumir llamando a las operaciones que estos expondrán. En la sección 5 se puede ver el proxy de la aplicación de servicio. Esta última será la clase que se asociará en las aplicaciones web que se ha creado y que realizara la interconexión con los clientes web. Esta clase será la que se utilizará desde una WebPart para poder consumir las operaciones.

* 1. Listado 5. – Clase ServicioGestionAplicacionProxy.cs
  2. namespace Msdn.ServiceApplication.AppServicioClientes
  3. {
  4. [Guid("2C009794-837C-4401-B3C9-E002C100E9A7")]
  5. [IisWebServiceApplicationProxyBackupBehavior]
  6. public class ServicioGestionAplicacionProxy : SPIisWebServiceApplicationProxy, IOrdenesDeVenta
  7. {
  8. private ChannelFactory<IOrdenesDeVenta> mChannelFactoryOrdenes;
  9. private object mChannelFactoryLock = new object();
  10. private String mEndpointConfigurationName;
  11. [Persisted]
  12. private SPServiceLoadBalancer mBalancer;
  13. [System.ComponentModel.EditorBrowsable(System.ComponentModel.EditorBrowsableState.Never)]
  14. public ServicioGestionAplicacionProxy()
  15. : base()
  16. {
  17. }
  18. public ServicioGestionAplicacionProxy(string pNombre, SPIisWebServiceProxy pServicioProxy, Uri pServicioEndPointUri)
  19. : base(pNombre, pServicioProxy, pServicioEndPointUri)
  20. {
  21. mBalancer = new SPRoundRobinServiceLoadBalancer(pServicioEndPointUri);
  22. }
  23. public override string TypeName
  24. {
  25. get { return ("MSDN - Servicio de Gestion Aplicacion Proxy"); }
  26. }
  27. public override void Provision()
  28. {
  29. mBalancer.Provision();
  30. base.Provision();
  31. this.Update();
  32. }
  33. internal Configuration Configuration
  34. {
  35. get
  36. {
  37. return OpenClientConfiguration(SPUtility.GetGenericSetupPath(@"WebClients\ServicioGestion"));
  38. }
  39. }
  40. internal SPServiceLoadBalancer LoadBalancer
  41. {
  42. get
  43. {
  44. return mBalancer;
  45. }
  46. }
  47. public override void Unprovision(bool deleteData)
  48. {
  49. mBalancer.Unprovision();
  50. base.Unprovision(deleteData);
  51. this.Update();
  52. }
  53. public List<Entidades.OrdenDeVenta> ObtenerTodasLasOrdenesDeVenta()
  54. {
  55. List<OrdenDeVenta> lRetorno = null;
  56. ExecuteOnChannel("ObtenerTodasLasOrdenesDeVenta", lChannel => lRetorno = lChannel.ObtenerTodasLasOrdenesDeVenta());
  57. return lRetorno;
  58. }
  59. public List<Entidades.OrdenDeVenta> ObtenerOrdenesDeVentaPorIdCliente(int pIdCliente)
  60. {
  61. throw new NotImplementedException();
  62. }
  63. public void IngresarOrdenDeVenta(Entidades.OrdenDeVenta pOrdenDeVenta)
  64. {
  65. throw new NotImplementedException();
  66. }
  67. public void ElimindarOrdenDeVenta(Entidades.OrdenDeVenta pOrdenDeVenta)
  68. {
  69. throw new NotImplementedException();
  70. }
  71. }
  72. }

Construido el servicio en una solución de SharePoint el próximo paso será instalarla en el servidor para que se despliegue en el mismo todos los componentes desarrollados y así poder empezar a usar el servicio de gestión. En la imagen figura 5 se puede ver el servicio disponible en la lista de servicios de cada servidor de SharePoint y donde se debe activar en el servidor que se desee tener el mismo corriendo ya que por defecto se instala deshabilitado.

* 1. Figura 5. - Servicio de gestión instalado
  2. 

Una vez iniciado el mismo se puede crear la aplicación de servicio en SharePoint para poder asociarla a la aplicación o aplicaciones web donde se desea usar, en la figura 6 se puede ver la aplicación de servicio creada con su correspondiente proxy, vale la pena destacar que si al crear la aplicación de servicio se deja la opción marcada que asocie esta aplicación al grupo por defecto de aplicaciones todas las aplicaciones web que tengan configurado la opción por defecto tendrán asociada dicha aplicación.

* 1. Figura 6. - Servicio de aplicación creado e iniciado
  2. 

Una vez el servicio esta iniciado y configurado se está en condiciones de poder consumir el mismo desde una WebPart en una aplicación web de SharePoint. En la sección 6 podemos ver cómo podemos obtener el proxy de nuestra aplicación de servicio usando el tipo.

* 1. Listado 6. - Obtención del servicio de aplicación por el tipo
  2. ServicioGestionAplicacionProxy lServicioGestion = (ServicioGestionAplicacionProxy)SPContext.Current.Site.WebApplication.ServiceApplicationProxyGroup.Proxies.OfType<ServicioGestionAplicacionProxy>().FirstOrDefault();
  3. lServicioGestion.ObtenerTodasLasOrdenesDeVenta();

Pero también se puede usar una expresión para obtener una referencia del proxy de aplicación por una de las propiedades que este objeto tiene, se debe recordar que una aplicación web puede tener uno o más servicios asociados, incluso del mismo tipo de servicio y deberemos poder tener acceso al servicio que nosotros deseamos utilizar, en la sección 7 se puede ver el código para obtener el servicio usando una expresión para la propiedad nombre.

* 1. Listado 7. - Obtención del servicio de aplicación mediante una expresión Linq
  2. IEnumerable<ServicioGestionAplicacionProxy> lServicioGestionEx = SPContext.Current.Site.WebApplication.ServiceApplicationProxyGroup.Proxies.OfType<ServicioGestionAplicacionProxy>().Where(x => x.Name.Contains("MSDN Aplicacion Sistema Gestion"));

Hasta ahora se ha visto cómo se puede obtener el servicio a través de la aplicación web donde se está ejecutando el código, pero se puede obtener el servicio a través de la clase “SPServiceContext” del modelo de objeto, como se puede ver en la Sección 8.

* 1. Listado 8. - Obtención del servicio de aplicación a través de la case SPServiceContext
  2. ServicioGestionAplicacionProxy lServicioGestion = (ServicioGestionAplicacionProxy)SPServiceContext.Current.GetDefaultProxy(typeof(ServicioGestionAplicacionProxy));
  3. lServicioGestion.ObtenerTodasLasOrdenesDeVenta();

Una forma que se tiene de invocar una operación es también llamando al método Invoke que expone la clase “ServicioGestionAplicacionProxy” creada, esto dependerá de la visibilidad con la que se creó el método. En este caso se ha creado dicho método público para poder consumirlo desde afuera del proyecto. Lo que se debe hacer es crear una clase cliente que encapsule la llamada a los servicios y que desde los componentes web que se han desarrollado, se brindé una forma sencilla de llamar a las operaciones expuestas por la aplicación de servicio. En la sección 9, se puede ver la definición de la clase cliente creada para el encapsulamiento de las llamadas.

* 1. Listado 9. - Clase cliente para encapsular las llamadas a las operaciones
  2. namespace Msdn.ServiceApplication.Servicios.UI
  3. {
  4. public class ServicioGestionOrdenesDeVentaCliente: IOrdenesDeVenta
  5. {
  6. private SPServiceContext mServiceContext;
  7. public ServicioGestionOrdenesDeVentaCliente(SPServiceContext pServiceContext)
  8. {
  9. if (pServiceContext == null)
  10. throw new ArgumentNullException("El contexto del servicio es nulo.");
  11. this.mServiceContext = pServiceContext;
  12. }
  13. public List<OrdenDeVenta> ObtenerTodasLasOrdenesDeVenta()
  14. {
  15. List<OrdenDeVenta> lResultado = null;
  16. ServicioGestionAplicacionProxy.Invoke(mServiceContext, proxy => lResultado = proxy.ObtenerTodasLasOrdenesDeVenta());
  17. return lResultado;
  18. }
  19. public List<Entidades.OrdenDeVenta> ObtenerOrdenesDeVentaPorIdCliente(int pIdCliente)
  20. {
  21. List<OrdenDeVenta> lResultado = null;
  22. ServicioGestionAplicacionProxy.Invoke(mServiceContext, proxy => lResultado = proxy.ObtenerOrdenesDeVentaPorIdCliente(pIdCliente));
  23. return lResultado;
  24. }
  25. public void IngresarOrdenDeVenta(Entidades.OrdenDeVenta pOrdenDeVenta)
  26. {
  27. ServicioGestionAplicacionProxy.Invoke(mServiceContext, proxy => proxy.IngresarOrdenDeVenta(pOrdenDeVenta));
  28. }
  29. public void ElimindarOrdenDeVenta(Entidades.OrdenDeVenta pOrdenDeVenta)
  30. {
  31. ServicioGestionAplicacionProxy.Invoke(mServiceContext, proxy => proxy.ElimindarOrdenDeVenta(pOrdenDeVenta));
  32. }
  33. }
  34. }

Por último en la sección 10 se puede cómo se debe consumir la clase cliente creada pasándole en el constructor una referencia de la instancia de la clase “SPServiceContext”. Esta clase cliente deberá tener las operaciones expuesta para que puedan ser invocadas, lo que permite poder tener un nivel más de control y validación de los datos antes de enviarlos a que sean ejecutados dentro de los procesos de negocio, de esta forma logramos crear una capa de servicios para la UI.

* 1. Listado 10. - Consumiendo servicio expuesto por una clase cliente
  2. ServicioGestionOrdenesDeVentaCliente lClinte = new ServicioGestionOrdenesDeVentaCliente(SPServiceContext.Current);
  3. lResultado = lClinte.ObtenerTodasLasOrdenesDeVenta();

Una vez construido el servicio y expuestas las operaciones de negocio, se construyeron los componentes web que consumirán los servicios correspondientes, en este ejemplo se han construido 3 WebParts para listar los datos desde la base de datos. En la siguiente imagen se puede ver la ejecución de la Webpart “Ordenes de Venta” que lista todas las ventas desde la base de datos.

* 1. Figura 7. - Componente web que lista las órdenes de venta
  2. 