Carrera: **Analista Programador**

Lenguaje de última generación

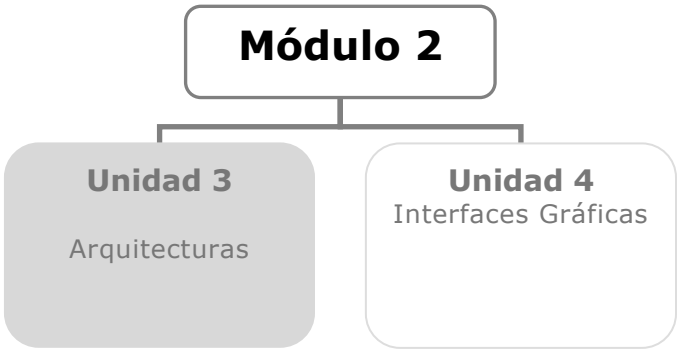
Módulo II

Arquitectura de SW.

Unidad 3

Arquitecturas

Docente y autor de contenidos: Ing. Mauricio Prinzo



Presentación

En esta unidad recordaremos algunos conceptos de los lenguajes de programación que son pilares fundamentales de las arquitecturas en el desarrollo de aplicaciones empresariales.

Por todo lo expresado hasta aquí es que esperamos que usted, a través del estudio de esta unidad, adquiera capacidad para:

- Acceder rápidamente a una base de datos relacional.
- Comprender las ventajas de trabajar en modo conectado o desconectado.
- Comprender las estructuraras de datos utilizadas en un lenguaje de programación
- Trabajar con estructuras planas de datos que faciliten el intercambio de datos.

A continuación, le presentamos un detalle de los contenidos y actividades que integran esta unidad. Usted deberá ir avanzando en el estudio y profundización de los diferentes temas, realizando las lecturas requeridas y elaborando las actividades propuestas, algunas de desarrollo individual y otras para resolver en colaboración con otros estudiantes y con su profesor tutor.

Contenidos y Actividades

1. Arquitecturas

- **Concepto de Arquitectura**
- **Como definir una arquitectura**
- **Tipos de arquitectura**
    - **Clásica cliente/servidor**
    - **Capas (Layers)**
        - **Modelo de 3 capas**
        - **Modelo de N-Capas**
    - **MVC**

Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)

Trabajo Práctico Sugerido

- Trabajo Práctico Nº 1: Programando en Capas.

  Este Trabajo práctico le permitirá una primera aproximación al proceso de identificación de objetos en un desarrollo real.

2. Conclusiones

Lectura Sugerida

- Deitel, Harvey M.; Deitel, Paul J.; Vidal Romero Elizondo, Alfonso(Traductor); y otros. **Cómo programar en C#.** 2a.ed.-- México, DF.
- Pearson Mayo, Joseph. **C# al descubierto**.-- Madrid : Pearson Educación, c2002. xxxi, 749 páginas

Trabajo colaborativo/Foro

- Debate sobre los fundamentos planteados en el módulo.

Para el estudio de estos contenidos usted deberá consultar la bibliografía que aquí se menciona:

BIBLIOGRAFÍA OBLIGATORIA

- Deitel, Harvey M.; Deitel, Paul J.; Vidal Romero Elizondo, Alfonso(Traductor); y otros. **Cómo programar en C#.** 2a.ed.-- México, DF.
- Pearson Mayo, Joseph. **C# al descubierto**.-- Madrid : Pearson Educación, c2002. xxxi, 749 páginas

Links a temas de interés

- http://Biblioteca.vaneduc.edu.ar/

El portal de la biblioteca universitaria con material y links de interés, además del asesoramiento de un referencista especializado.

.

1. Arquitectura de software

En los inicios de la informática, la programación se consideraba un arte y se desarrollaba como tal debido a la dificultad que entrañaba para la mayoría de las personas, pero con el tiempo se han ido descubriendo y desarrollando formas y guías generales, con base a las cuales se puedan resolver los problemas.

A estas, se les ha denominado arquitectura de software, porque, a semejanza de los planos de un edificio o construcción, estas indican la estructura, funcionamiento e interacción entre las partes del software.

Por los años 1960 comenzaba a sonar el concepto de arquitectura de software en los círculos de investigación (por ejemplo, por Edsger Dijkstra) y como todo lo que incursiona toma popularidad más tarde, esta tardo en 30 años en ser popular.

En 1990 se toma interés de la disciplina de la Ing. del software.

Los motivos, el uso de estructuras de SW:

– Complejidad creciente de aplicaciones.

– Sistemas distribuidos (segunda coordenada de complejidad).

– Sistemas abiertos y basados en componentes (tercera coordenada de complejidad).

1.1. Concepto de Arquitectura

Cuando hablamos de arquitectura tenemos que tener en cuenta ciertos conceptos:

Vista estructural de alto nivel: La arquitectura de un sistema describe en un alto nivel de abstracción como se encuentra estructurado el mismo, que servicios expone tanto interna como externamente y como se comunican dichos servicios entre sí.

Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)

Se concentra en requerimientos no funcionales: Los requerimientos no funcionales definen la arquitectura en gran parte. Por ejemplo, "Escalabilidad" … si armo una aplicación monolítica tengo muy pocas probabilidades de poder distribuirla en más de un nodo de procesamiento a futuro….

Balance: una estructura tiene q estar balanceada.

Es esencial para el éxito o fracaso de un proyecto: Una arquitectura errónea podría provocar como producto un sistema difícil de mantener, escalar y que no logre los requerimientos de disponibilidad y seguridad necesarios…. Entre otros problemas

1.2 ¿Cómo definir una Arquitectura de SW?

Para poder definir una arquitectura de SW tenemos que tener en cuenta:

Identificar los requerimientos no funcionales: Por ejemplo, "Disponibilidad" si tengo que asegurar un alto grado de disponibilidad de una aplicación es muy probable que tenga que distribuir entre varios servidores para que esté disponible y no tenga problemas de rendimiento… para ello tengo que optar por una arquitectura distribuida.

Identificar interacciones con otros sistemas: cómo se va a relacionar con otros sistemas y a través de que tecnologías

Planificar la evolución del sistema, identificando las partes mutables e inmutables de la misma, así como los costos de los posibles cambios.

(Con una estructura se define para poder realizar cambios sin que el sistema sufra inconvenientes)

… por ejemplo… ¿El sistema algún día tendrá que ser web?

-Definir las tecnologías a implementar en la solución: Indefectiblemente las tecnologías a utilizar van a impactar en nuestras decisiones sobre la arquitectura del sistema…. ¿podemos hacer una aplicación orientada a servicios con COBOL?.... bueno, aunque no lo crean si

This document is available free of charge on studocu
Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)

1.3 Tipos de Arquitectura

Hay muchos tipos de arquitectura, pero vamos a ver los más utilizados.

1.3.1 Cliente/Servidor

En una arquitectura Cliente/Servidor los datos son compartidos o almacenados en un servidor central.

Se pueden compartir archivos e impresoras, pero la entrada, procesamiento y salida de la información se continúa realizando en la misma PC del cliente.

1.3.2 Capas (Layers)

Todo sistema se encuentra compuesto por un conjunto de subsistemas

Estos subsistemas a su vez pueden ser agrupados en una capa lógica y también pueden ser divididos en varias capas lógicas (se recuerda que un subsistema es un sistema que forma parte de un sistema mayor y por lo tanto tiene todas las características de un sistema).

Las capas lógicas son agrupaciones de subsistemas (o componentes) que exponen una funcionalidad de alto nivel (por ejemplo, la capa lógica de negocios que puede estar conformada por el subsistema de control de inventario, subsistema de control de precios, etc).

En el contexto de la arquitectura lógica los conceptos de paquete, servicio y capa son sinónimo.

Las capas lógicas se comunican entre si por medio de interfaces (interfaz de subsistema o servicio) y cada interfaz de cada capa expone los servicios de la misma.

Esta visión de interfaces permite que cada capa sea vista como una "caja negra" de la cual se conoce como solicitar un servicio y que resultados produce tal solicitud, pero se ignora totalmente como trabaja internamente la capa para producir dichos resultados. De esta forma se disminuye el aco-

Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)

plamiento entre capas logrando que cada una dependa únicamente de las interfaces de los servicios que utiliza de otra capa.

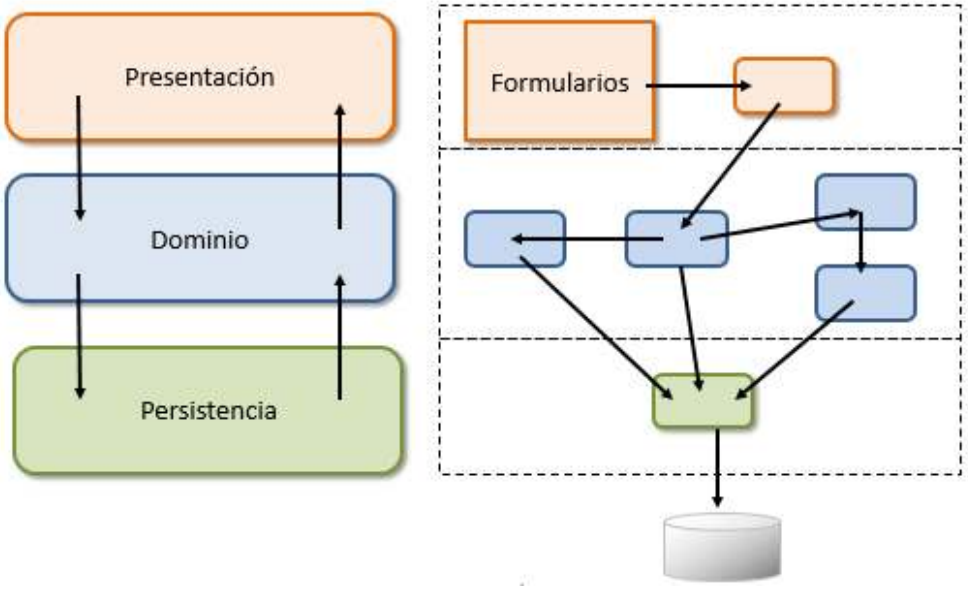
Las capas pueden brindar sus servicios de forma transversal o vertical

Como nombras las Capas (Layers) se ocupan de la división lógica de componentes y funcionalidad, y no tienen en cuenta la localización física de componentes en diferentes servidores o en diferentes lugares.

Por el contrario, los Niveles (Tiers) se ocupan de la distribución física de componentes y funcionalidad en servidores separados, teniendo en cuenta topología de redes y localizaciones remotas. Aunque tanto las Capas (Layers) como los Niveles (Tiers) usan conjuntos similares de nombres (presentación, servicios, negocio y datos), es importante no confundirlos y recordar que solo los Niveles (Tiers) implican una separación física. Se suele utilizar el término "Tier" refiriéndonos a patrones de distribución física como "2 Tier", "3-Tier" y "N-Tier".

1.3.1.1 Arquitectura en 3 Capas

La arquitectura clásica es la de 3 capas



This document is available free of charge on studocu
Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)

Presentación: Concretamente, es la encargada de "decidir" como deben mostrar los datos en las GUI.

Capturar los inputs del usuario

Debido a que el usuario interactuará con una GUI, la capa de presentación es responsable de capturar las entradas del usuario e invocar a los servicios necesarios para poder llevar a cabo las funciones requeridas por el usuario en ese momento.

Dominio o Negocio: La capa de dominio contiene la lógica del dominio (las reglas de negocio en el caso del proyecto actual

Responsabilidades: Exponer los servicios de la lógica de negocios.

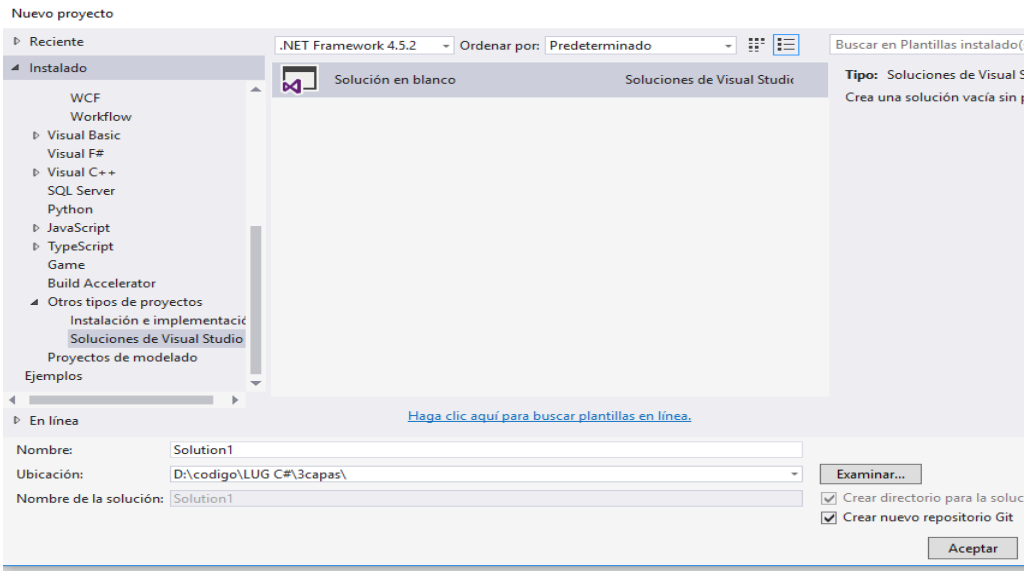
En la capa de dominio se debe implementar toda la lógica que permite llevar a cabo los servicios de la lógica de negocios para que puedan ser utilizados por el software cliente. El software cliente por ejemplo puede ser una GUI Web, una GUI Desktop o un Web Service.

Persistencia: La capa de persistencia es la capa que se encarga de que los objetos persistan entre ejecuciones del sistema en un mecanismo de persistencia.

El mecanismo de persistencia puede ser una base de datos relacional, una base de datos orientada a objetos, archivos XML, etc.

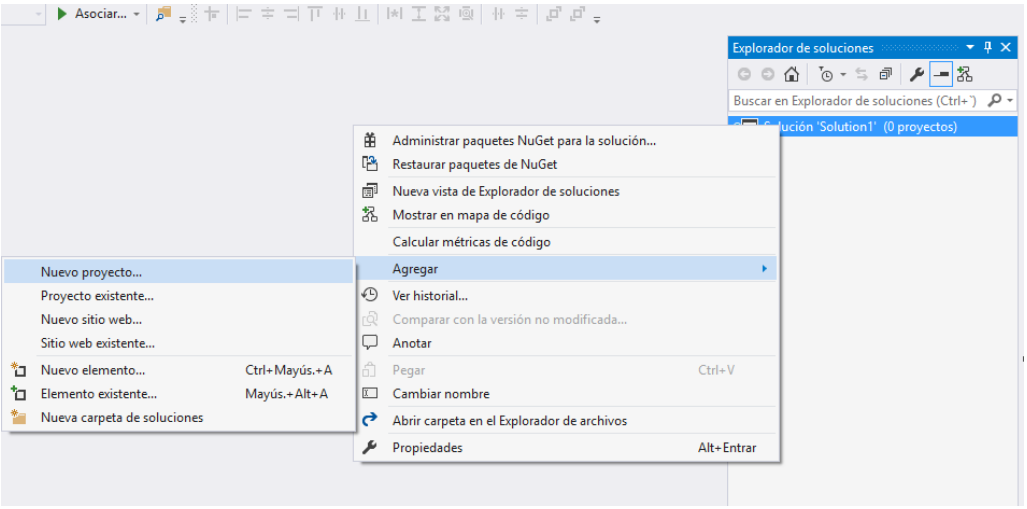
Veamos cómo realizar un proyecto en 3 capas con .NET

Primero vamos a crear un proyecto y seleccionamos la solución en blanco.

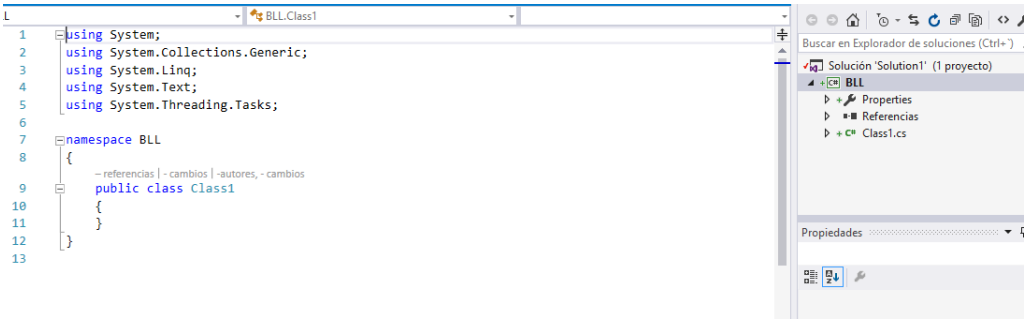


Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)

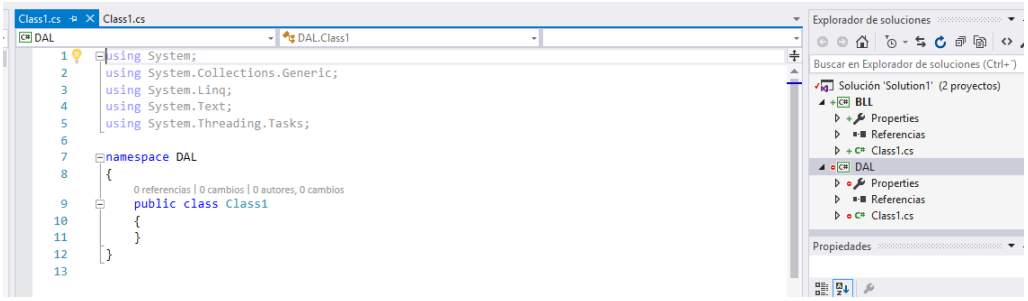
Segundo, no paramos sobre la solucion creada, y boton derecho click, agregar nuevo proyecto de clases.



El proyecto lo llamo BLL(Busineds Logicla Layer), refiriendome a la capa de negocio

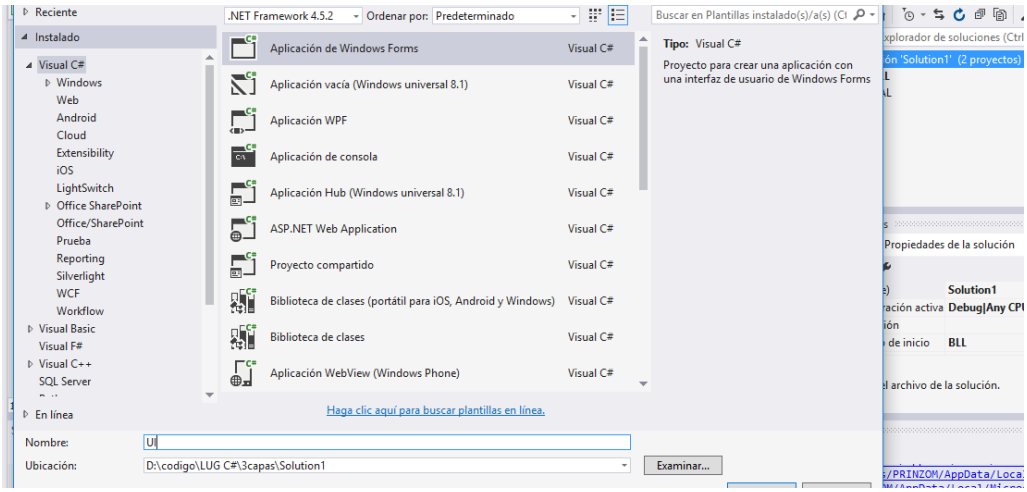


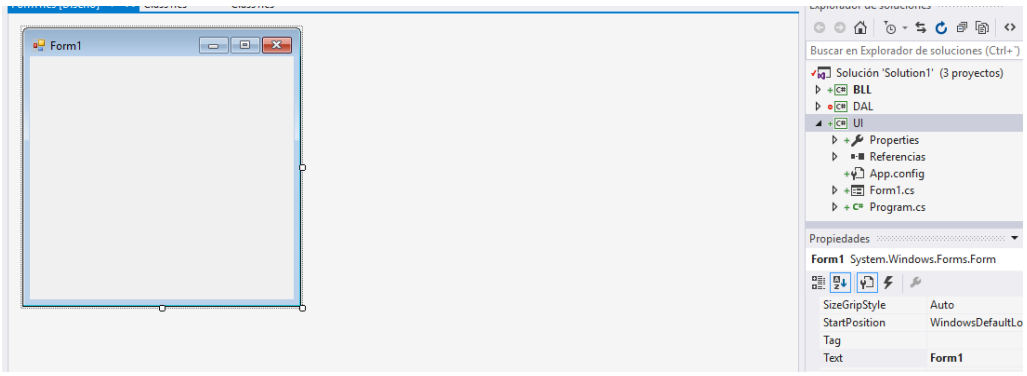
De la misma manera creo otro llamado DAL(Data acces Layer), correspondiente a la capa de persistencia de datos.



This document is available free of charge on studocu
Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)

Luego creo la capa de presentación, ósea la UI (User interface)



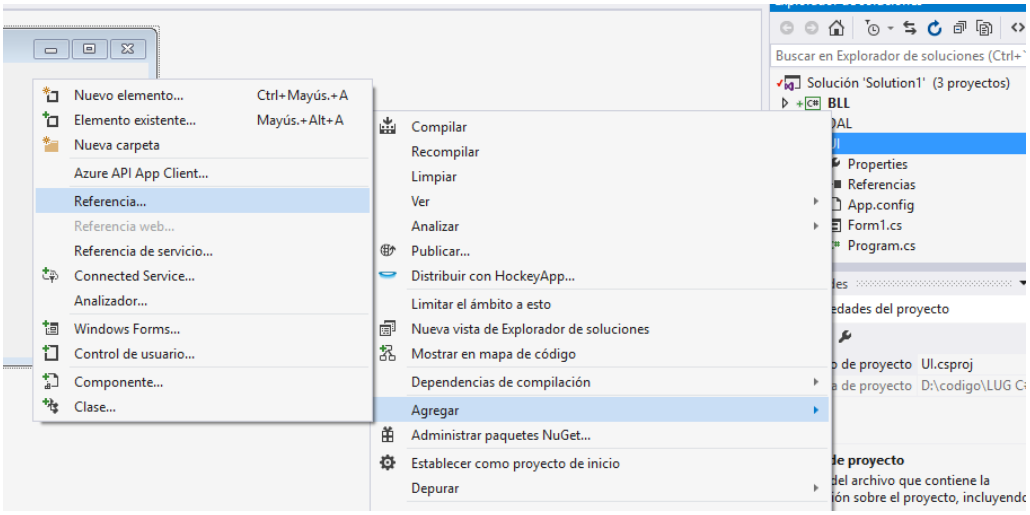


De esta manera, tengo las 3 capas logicas, lo que faltaria es como relacionarlas.

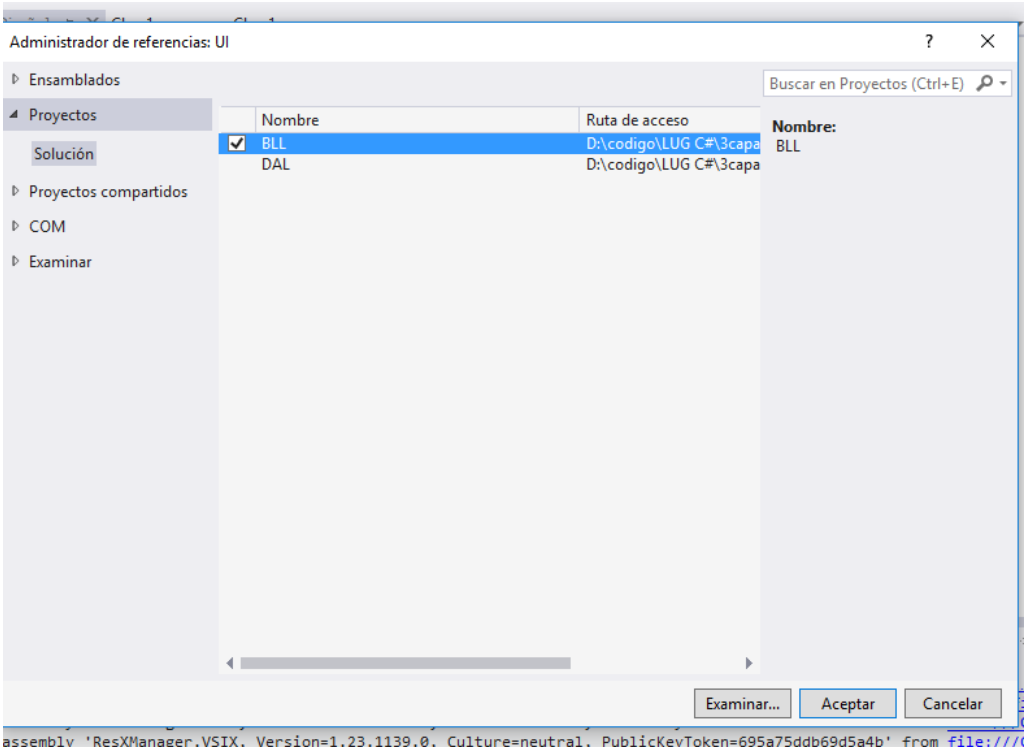
La relacion tiene que ser de la capa que requiere informacion de la otra capa, en el modelo de 3 capas seria de la UI con la BLL y la BLL con la DAL, para ello le agregamos una referencia de la siguiente manera

Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)

Me paro sobre el proyecto, en este caso la UI, y voy a la opcion de agregar, luego a referencia

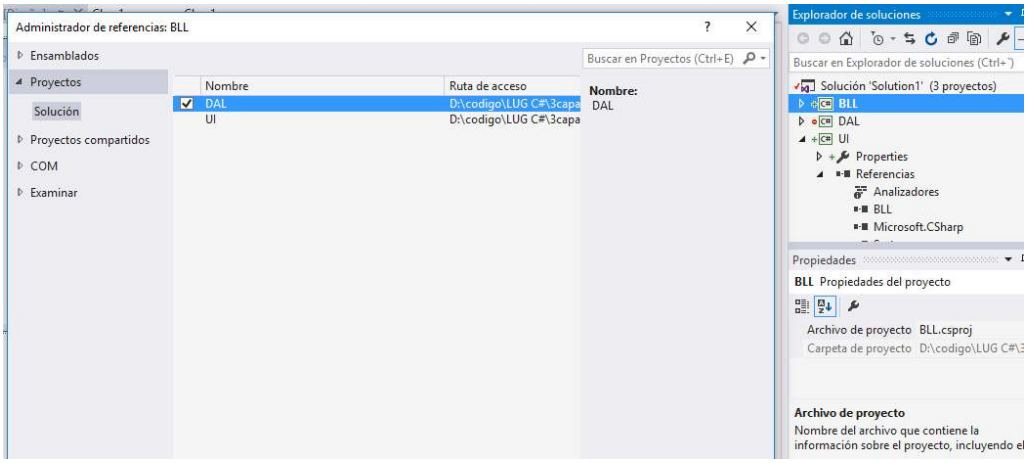


Nos van aparecer los proyectos a los cuales se pueden hacer referencia marcamos la BLL



This document is available free of charge on studocu
Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)

Hago lo mismo con la BLL y la DAL



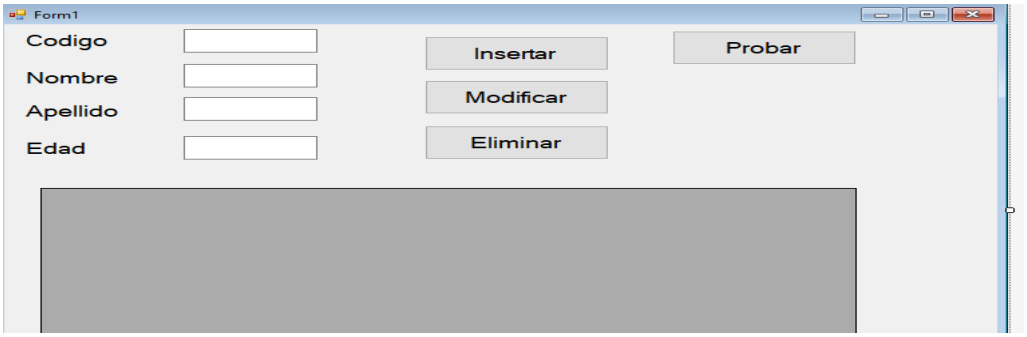
Si bien tenermos referenciadas las capas, para poder utilizarlas hay que realizar el using de dicha capa que estoy referenciando.

```
//de la UI a la BLL
using BLL;

// de BLL a DAL
using DAL;
```

Luego en cada capa escribimos el código que corresponda.

Ejemplo para el siguiente ABM

En la UI



Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)

```
//Código del botón Insertar de la UI
private void buttonINsertar_Click(object sender, EventArgs e)
        {
       //instancio la clase Alumno  de la BLL para pasar los datos
            Alumno oAlu = new Alumno();

            if (textBox4.Text != "")
            { oAlu.codigo = Convert.ToInt32(textBox4.Text); }
            else { oAlu.codigo = 0; }

            oAlu.Nombre = textBox1.Text;
            oAlu.Apellido = textBox2.Text;
            oAlu.Edad = Convert.ToInt32(textBox3.Text);

  //llamo al metodo de alta de Alumno de la BLL y paso el mismo ob-
jeto con los datos
            oAlu.ALta(oAlu);
            //metodo para cargar el gridView
            Enlazar();
            //metodo que limpia los textbox
            Limpiar();
        }
```

En la BLL escribimos el código para dar de alta al alumno

```
  public void ALta(Alumno oAlu)

  {   //acceso es la clase de la DAL
            acceso oDatos = new acceso();

            string sql = "Insert into ALUMNO  (nombre, apellido,
edad) values ('" + oAlu.Nombre + "','" + oAlu.Apellido + "'," +
oAlu.Edad + " ) ";

            oDatos.Escribir(sql);
      }
```

En la DAL envió al método escribir que es genérico

This document is available free of charge on studocu
Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)

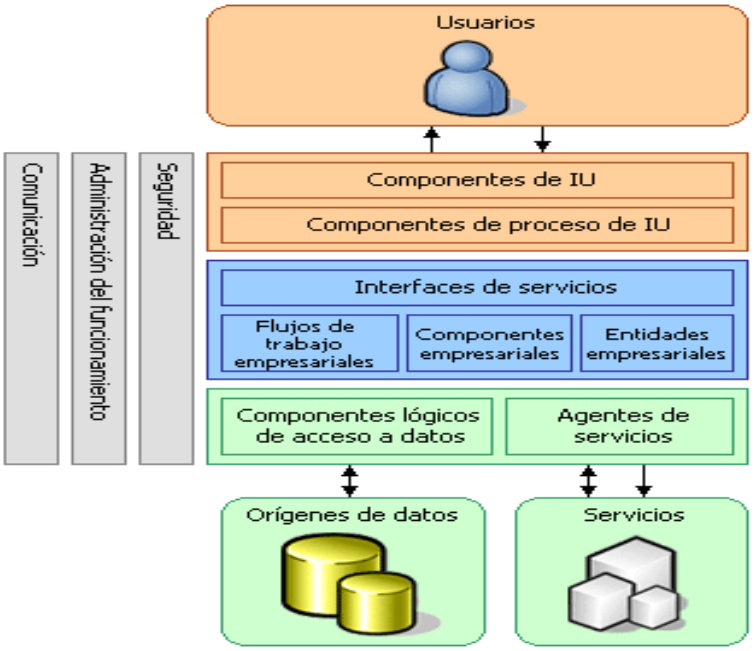
```
  public class acceso
    {   //declaro el objeto del tipo conction
        private SqlConnection conexion;

        //creo el método Abrir, le paso la cadena de conexión para saber
la dirección de la BD y la abro con la propiedad Open
        public void Abrir()
        {  conexion = new SqlConnection();

            conexion.ConnectionString = @"Data
Source=.\SQLEXPRESS;Initial Catalog=MiBase;Integrated Security=True";
            conexion.Open();
        }
        //creo el método Cerrar la conexión y limpio la misma de memoria
con el Dispose.
        public void Cerrar()
        {
            conexion.Close();
            conexion.Dispose();
            conexion = null;
            GC.Collect();
        }
//para ver si la conexión está abierta
        public string TestConnection()
        {
            Abrir();
            //Disconnect();
            if (conexion.State == ConnectionState.Open)
            {   return "Conexion Opened";    }
            else
            {  return "Conexion Not opened";  }
        }
        //creo una función genérica para escribir en la BD, solo pasando
la consulta a impactar en la BD
        public int Escribir(string SQL)
        {
            int filasAfectadas = 0;
            Abrir();
            SqlCommand cmd = new SqlCommand();
            cmd.CommandType = CommandType.Text;
            cmd.Connection = conexion;
            cmd.CommandText = SQL;
            try
            {   filasAfectadas = cmd.ExecuteNonQuery();   }
            catch
            {   filasAfectadas = -1;            }

            Cerrar();
            return filasAfectadas;
        }
    }
```


Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)

Trabajando en capas genero independencia de código, puedo modificar una capa sin alterar otra.

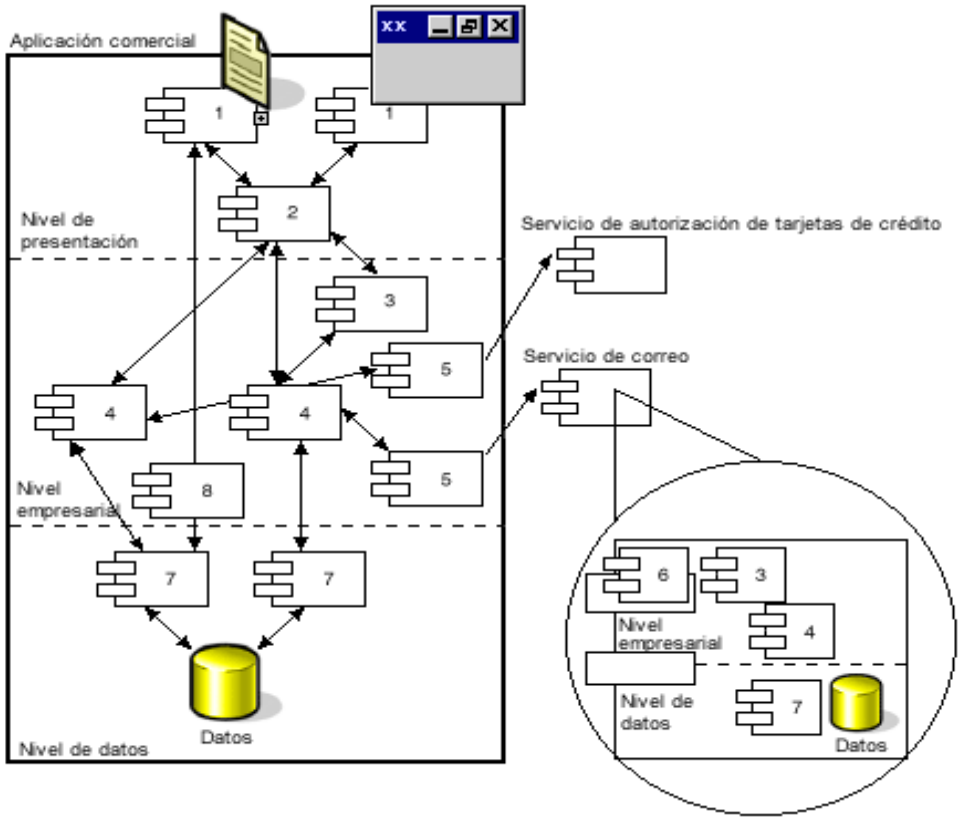
1.3.1.2 Arquitectura en N-Capas

La arquitectura de n capas es la segmentación de una aplicación en partes y la distribución de esas partes a lo largo de una red.
La ventaja de utilizar una arquitectura de n capas está en que la capacidad de ejecutar, mantener y mejorar aplicaciones de gran escala aumenta considerablemente.

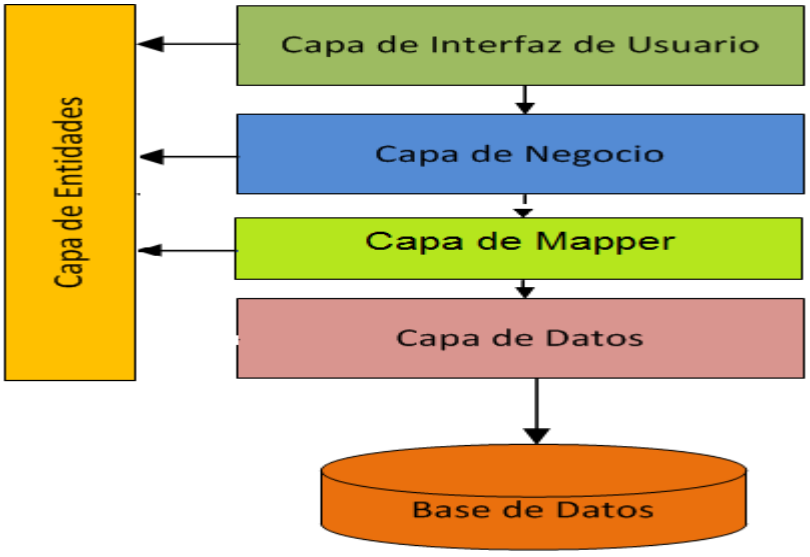


This document is available free of charge on studocu
Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)

En la figura de la abajo podemos ver que una capa no solo tiene 1 componente, una capa puede tener varios componentes.



Un ejemplo de N-capas, siendo N=5.



Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)

Interfaz de Usuario (UI -User Interfaces): Es la capa de presentación, con la que interactúan los usuarios.

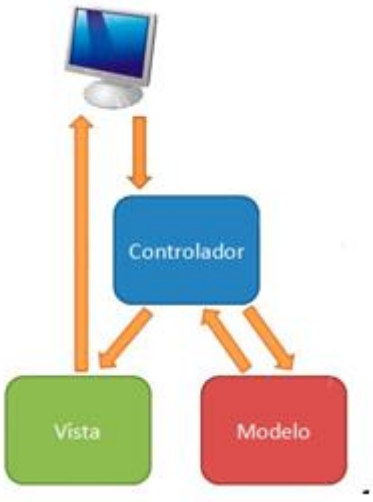
Entidades: se utiliza para la comunicación o medio de transporte entre las demás capas para llevar y traer datos, regularmente a través de parámetros (solamente las propiedades de las clases)

Negocio (BLL -Business Logic Layer): Esta capa contiene toda la lógica del negocio.

Mapper (MPP): Capa que convierte objetos a datos relacionales y viceversa. Posee la lógica para la transformación.

Datos (DAL -Data Access Layer): Es la capa encargada de la persistencia en la base de datos

1.3.3 MVC



Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Se trata de un modelo muy maduro y que ha demostrado su validez a lo largo de los años en todo tipo de aplicaciones, y sobre multitud de lenguajes y plataformas de desarrollo.

This document is available free of charge on studocu
Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)

El **Modelo** que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.

La **Vista**, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.

El **Controlador**, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

El modelo es el responsable de acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.
Define las reglas de negocio (la funcionalidad del sistema). Un ejemplo de regla puede ser: "Si la mercancía pedida no está en el almacén, consultar el tiempo de entrega estándar del proveedor".
Lleva un registro de las vistas y controladores del sistema.
Si estamos ante un modelo activo, notificará a las vistas los cambios que en los datos pueda producir un agente externo (por ejemplo, un fichero por lotes que actualiza los datos, un temporizador que desencadena una inserción, etc.).

El controlador es responsable de recibir los eventos de entrada (un clic, un cambio en un campo de texto, etc.).
Contiene reglas de gestión de eventos, del tipo "SI Evento Z, entonces Acción W". Estas acciones pueden suponer peticiones al modelo o a las vistas. Una de estas peticiones a las vistas puede ser una llamada al método "Actualizar()". Una petición al modelo puede ser "Obtener_tiempo_de_entrega (nueva_orden_de_venta)".

Las vistas son responsables de recibir datos del modelo y los muestra al usuario.
Tienen un registro de su controlador asociado (normalmente porque además lo instancia).
Pueden dar el servicio de "Actualización()", para que sea invocado por el controlador o por el modelo (cuando es un modelo activo que informa de los cambios en los datos producidos por otros agentes).

El flujo que sigue el control generalmente es el siguiente:

1) El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)

2) El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.

Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)

3) El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.

4) El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, se podría utilizar el patrón Observador para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.

5) La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

Trabajo Práctico Sugerido

- Trabajo Práctico Nº 1: Realice un ABM en 3 capas

Cierre de la unidad

Trabajo colaborativo/Foro

- Debate sobre los beneficios de usar arquitectura de 3 o N capas en nuestros desarrollos

Fin de la unidad 3

This document is available free of charge on studocu
Descargado por Gerardo Tordoya (zherar7ordoya@outlook.com)