TRANSCRIBIR

Resolución Jefatural N° 1591 -2011-ED

Lima, 30 MAY 2011

Vistos, el Expediente Nº 111336-2011 y demás actuados, y

CONSIDERANDO:

Que, el artículo 20° del Reglamento de Organización y Funciones del Ministerio de Educación, aprobado mediante el Decreto Supremo N° 006-2006-ED, señala que la Oficina de Informática es responsable de establecer las políticas, normas y estándares, así como conducir el uso de recursos informáticos en el Sector Educación;

Que, el literal "h" del artículo 20° del mencionado dispositivo establece que es función de la Oficina de Informática el formular y proponer las políticas destinadas a estandarizar las herramientas informáticas, procesos y niveles de calidad de la gestión, productos y servicios desarrollados por la Oficina de Informática;

Que, dentro de este marco normativo, se propone la aprobación de una guía para el desarrollo de aplicaciones distribuidas, instrumento cuyo propósito es el definir un conjunto de criterios en nomenclatura, orden y organización de la programación de código fuente y demás componentes de software que forman parte de la solución de los proyectos;

Que, en ese sentido, deviene en necesaria la aprobación de la mencionada guía, a fin de mejorar los estándares en los procesos de programación informática desarrollados en el Ministerio de Educación;

De conformidad con lo establecido en el Decreto Ley Nº 25762 modificado por la Ley Nº 26510; el Decreto Supremo Nº 006-2006-ED y sus normas modificatorias;

SE RESUELVE:

Artículo Primero.- Aprobar el documento denominado *"Guía para el Desarrollo de Aplicaciones Distribuidas"*, cuyo texto en Anexo adjunto forma parte de la presente Resolución.

Artículo Segundo.- Disponer la publicación de la presente Resolución y su Anexo en el Portal Institucional del Ministerio de Educación (www.minedu.gob.pe).

Regístrese y comuníquese,

MINISTERIO DE EDUCACIÓN - Oficina de Informática - JEFE

MANUEL COK APARCANA
Jefe de la Oficina de Informática

1591 -2011-ED

MINISTERIO DE EDUCACIÓN
REPUBLICA DEL PERÚ

Secretaría de Planificación Estratégica
Oficina de Informática

Guía para el desarrollo de aplicaciones distribuidas.

Versión 1.0

1591-2011-ED

Contenido

1591-2011-ED

1. Introducción

Este documento ofrece un marco de buenas prácticas, lineamientos y guías para el desarrollo de aplicaciones distribuidas y es independiente de la metodología de desarrollo de sistemas seleccionada.

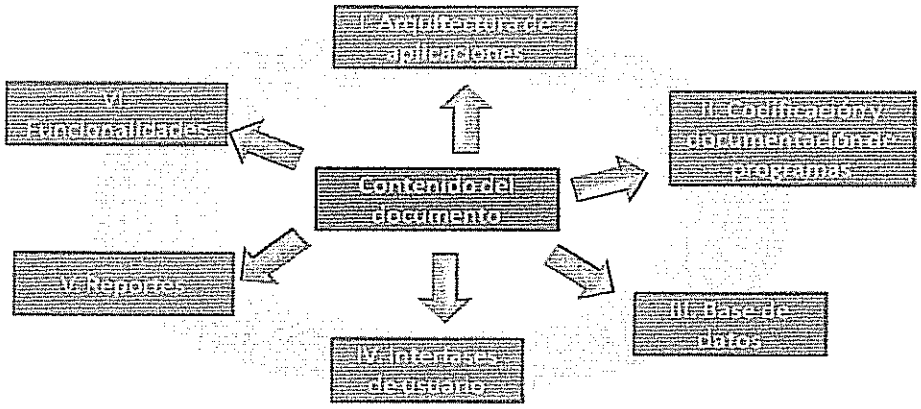
Se entiende por una aplicación distribuida a aquella que está constituida por distintos componentes que se pueden encontrar en entornos separados, normalmente en diferentes plataformas conectadas a través de una red. Son aplicaciones construidas por partes a las cuales se les asigna un conjunto específico de responsabilidades.

También es propósito de este documento el definir un conjunto de criterios en nomenclatura, orden y organización de la programación de código fuente y demás componentes de software (clases, formularios, páginas, etc.) que forman parte de la solución de los proyectos. Los miembros del equipo de desarrollo de serán los responsables, de acuerdo a su función, de cumplir y hacer cumplir los estándares aquí planteados.

Este documento consta de cinco capítulos organizados de la siguiente manera: El primer capítulo trata sobre la arquitectura de las aplicaciones desarrolladas en el Ministerio de Educación. El segundo capítulo se refiere a aspectos de la codificación de programas.

El tercer capítulo aborda los aspectos relevantes del manejo de base de datos. El cuarto capítulo aborda las especificaciones de las interfases de usuario y el quinto se refiere a las especificaciones de reportes.

Finalmente, el sexto capítulo está referido a las funcionalidades que deben tenerse en cuenta al momento de desarrollar una aplicación.



Este documento pretende ser lo suficientemente detallado para servir como apoyo al momento iniciar un nuevo proyecto de desarrollo o mantenimiento de sistemas de información y lo suficientemente genérico para no comprometer la seguridad de los sistemas de información evitando exponer detalles de la construcción y los componentes involucrados. También es entendido que unas guías muy detalladas no fomentan la creatividad de los desarrolladores y arquitectos.

La Oficina de Informática ha adoptado desde inicios del 2008 la tecnología .Net como herramienta predominante en el desarrollo de sus aplicaciones, decisión a la cual se llegó a través de un minucioso análisis de las alternativas disponibles. Este análisis quedó plasmado en el Informe Previo de Evaluación de Software N° 031: Herramientas estándar de desarrollo de aplicaciones OFIN – MED, el mismo que ha estado publicado desde su emisión en el portal del MINEDU en el URL:

http://www.minedu.gob.pe/transparencia/2011/datos/InformesTecnicosSoftware.php

MINISTERIO DE EDUCACION VISACION Oficina de Informática

MINISTERIO DE EDUCACION VISACION Oficina de Informática

MINISTERIO DE EDUCACION VISACION Oficina de Informática

2. Generalidades del documento

1591 -2011-ED

2.1. Alcance

El documento tendrá alcance para todos aquellos especialistas informáticos que participen directa o indirectamente, a tiempo parcial o completo en el desarrollo y/o mantenimiento de cualquiera de los aplicativos que forman parte del Ministerio de Educación.

Las recomendaciones y ejemplos presentados en este documento abarcan las versiones del Framework 3.5. El lenguaje utilizado para la presentación de ejemplos es el C# que corresponde al estándar establecido por la Oficina de Informática.

Asimismo, las buenas prácticas abarcarán tecnologías como: ASP.NET, WCF, ASP.NET y Ajax.

2.2. Convenciones

Las convenciones mostradas líneas abajo permitirán que las recomendaciones y líneas guía puedan ser usadas de manera apropiada y con el énfasis correspondiente.

En seguida algunas de las convenciones usadas a través del documento, con respecto a las palabra clave:

Siempre	Enfatiza que la regla debe ser cumplir.
Nunca	Enfatiza que la acción nunca se debe de ejecutar.
Evitar	Enfatiza que la acción debe de ser prevenida, pero algunas excepciones pueden existir.
Tratar	Enfatiza que la regla debe de cumplirse siempre y cuando sea posible y sea adecuado.
Ejemplo	Precede el texto usado para ilustrar la regla o recomendación.
Razón	Explica el propósito detrás de la regla o recomendación.

MINISTERIO DE EDUCACION - VISACION - Oficina de Informática

MINISTERIO DE EDUCACION - VISACION - Oficina de Informática

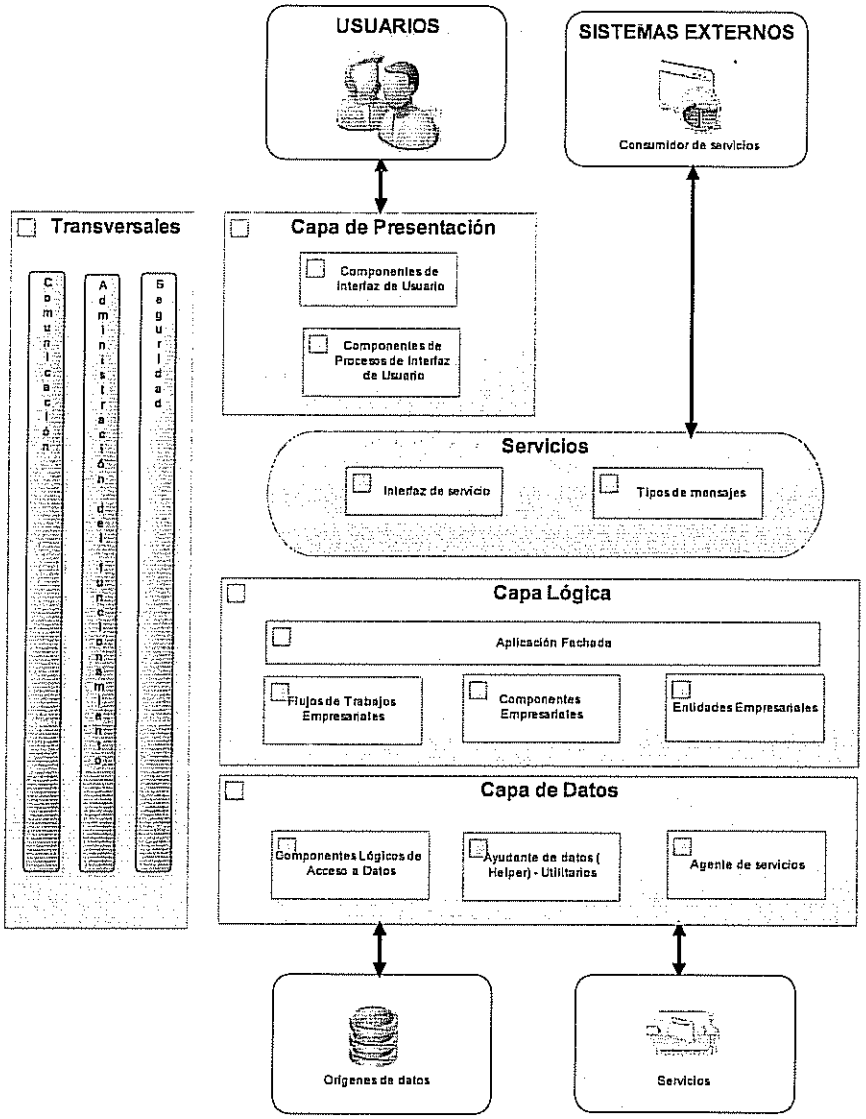
MINISTERIO DE EDUCACION - VISACION - Oficina de Informática

1591-2011-ED

CAPÍTULO I

3. Arquitectura de aplicaciones

3.1. Diagrama Lógico



3.2. Diagrama Físico

El diagrama físico de la arquitectura de aplicaciones (que es una extensión del diagrama lógico) no forma parte de este documento porque expone el detalle de cómo se vienen desplegando las aplicaciones en los ambientes de desarrollo de la Oficina de Informática.

1591 -2011-ED

CAPÍTULO II

4. Lineamientos para la codificación de programas

4.1. Convenciones de Nombres

En esta sección se describirá la forma apropiada de nombrar los distintos elementos y objetos que utilizamos en desarrollar un proyecto de software. Los temas a tratar serán: forma correcta de nombrar proyectos, archivos de código fuente, e identificadores incluyendo campos, variables, propiedades, métodos, parámetros, clases, interfases y espacios de nombres.

4.1.1. Líneas Generales

a. Siempre usar nombres con estilo de escritura PascalCase.
b. Evitar nombres totalmente en MAYÚSCULAS o en minúsculas. Nombres de una sola palabra serán totalmente en minúsculas. Las constantes siempre se escribirán en mayúsculas.
c. Nunca cree espacios de nombres, clases, métodos, propiedades, campos o parámetros que varían solamente por su capitalización.
d. Nunca usar nombres que comiencen con caracteres numéricos.
e. Siempre escoja nombres específicos y con significado en si mismos.
f. Variables y propiedades deben describir la entidad que representan no el tipo o tamaño.
g. Nunca usar notación Húngara para la declaración de variables.

Ejemplo: strNombre o chrTipoRol

h. Evitar el uso de abreviaturas a menos que el nombre completo sea excesivo.
i. Evitar abreviaturas que su longitud sea mayor a 5 caracteres.
j. Cualquier abreviatura debe ser totalmente conocida y aceptada por el equipo de desarrollo, y a su vez debe ser plasmada en un diccionario de abreviaturas de desarrollo.
k. Usar mayúsculas en caso de abreviaciones de 2 o 3 letras, y estilo de escritura PascalCase para abreviaturas más largas.
l. Nunca usar palabras reservadas para nombres.
m. Evitar conflictos de nombres con los espacios de nombres o tipos existentes en el .NET Framework.
n. Evitar añadir redundancia en prefijos o sufijos de los identificadores. Por ejemplo:

```
// Mal Escrito
public enum EstadoDocenteEnum { ... }
public class CAdministracionNotas { ... }
public struct UbicacionStruct { ... }
```

o. Nunca incluir el nombre de la clase a los nombres de las propiedades.

Ejemplo: Docente.Nombres **NO** Docente.NombresDocente

p. Tratar de utilizar los siguientes prefijos en las variables y propiedades booleanas, "Puede", "Es" o "Tiene".

4.1.1.1. Prefijo de controles en .Net

A continuación se indican los prefijos de los principales controles utilizados en el desarrollo de aplicaciones:

Controles de ASP.NET

Controles	Prefijo (minúscula)
Standard	
Label	lbl
TextBox	txt
Button	btn

1591 -2011-ED

LinkButton	lbtn
ImageButton	ibtn
HyperLink	lnk
DropDownList	ddl
ListBox	lst
CheckBox	chk
CheckBoxList	chkl
RadioButton	rb
RadioButtonList	rbl
Image	img
HiddenField	hdf
Calendar	cal
FileUpload	fu
MultiView	mv
Panel	pnl
PlaceHolder	ph
View	v
Data	
GridView	gv
DataList	dl
DetailsView	dv
ListView	lv
FormView	fv
Repeater	rt
DataPager	dp
Validación	
RequiredFieldValidator	rfv
RangeValidator	rv
RegularExpressionValidator	rev
CompareValidator	cv
CustomValidator	cuv
ValidatorSummary	vs
Navegación	
SiteMapPath	sitemap
Menu	me
TreeView	tv
AJAX	
ScriptManager	sm
ScriptManagerProxy	smp
Timer	tm
UpdatePanel	upnl
UpdateProgress	upr
Reportes	
MicrosoftReportViewer	rpt

4.1.1.2. Uso de nombres y sintaxis

Identificador	Convención de Nombramiento
Nombre de la Solución	Utilizar estilo de escritura PascalCase. El nombre de la solución siempre debe seguir el siguiente formato: [NombreCompañia].[NombreProjecto] Ejemplo: MED.MiProyecto
Archivo de Proyecto	Utilizar estilo de escritura PascalCasel. El nombre del proyecto siempre debe seguir el siguiente formato: [NombreCompañia].[NombreProjecto].[NombreCapa] El nombre del espacio de nombres y el ensamblado debe ser el mismo también. Ejemplo: MED. MiProyecto.Datos
Carpetas dentro de los Proyectos	Utilizar estilo de escritura PascalCasel. En caso de que el proyecto cuente con varios módulos, se recomienda el uso de carpetas dentro del proyecto, para ello siempre se debe tener en cuenta el siguiente formato: [NombreCompañia].[NombreProjecto].[NombreCapa].[NombreMódulo] Ejemplo: MED.MiProyecto.Datos.Administrador **Nota:** El namespace generado para los archivos dentro de las carpetas debe

	referenciar solo hasta el nombre del proyecto.
Archivo Fuente	Utilizar estilo de escritura PascalCase. Siempre debe coincidir el nombre de la clase y el nombre del archivo. Se debe evitar incluir más de una clase, enumerador (global) o delegado (global) por archivo. **Nota:** Si se utiliza archivos con múltiples clases, enumeradores o delegados entonces usar un nombre descriptivo para el archivo. Ejemplo: MiClase.cs => public class MiClase
Archivo de Recurso	Utilizar estilo de escritura PascalCase. En lo posible utilizar un nombre que describa la información que contenga el archivo.
Espacio de Nombre	Estilo de Escritura PascalCase. Este debe coincidir con el nombre del ensamblado/proyecto. Ejemplo: namespace MED.MiProyecto.Datos
Clase o Estructura	Utilizar estilo de escritura PascalCase. Usar un sustantivo o frase de sustantivos para el nombre de la clase. Agregar un sufijo apropiado cuando se nombre sub clases. Ejemplo: private class MiClase { ... } public class ClienteEventArgs : EventArgs { ... } private struct PropiedadesAplicacion { ... } Para el caso específico de las capas a utilizar, siempre seguir el siguiente formato: - Para la capa de entidades de negocio se utilizará el siguiente prefijo: BE[NombreEntidad] - Para la capa de datos se utilizará el siguiente prefijo: DA[NombreEntidad] - Para la capa de lógica de negocios se utilizará el siguiente prefijo: BL[NombreEntidad] - Para los servicios se utilizará el siguiente sufijo: [Proyecto/Módulo]Services Ejemplo: public class BECliente { ... } public class DACliente { ... } public class BLCliente { ... } public class PronafcapServices { ... }
Interfase	Utilizar estilo de escritura PascalCase. Siempre agregar el prefijo "I" Ejemplo: interface ICliente { ... }
Método	Utilizar estilo de escritura PascalCase. Tratar de usar un verbo o par verbo-objeto. Ejemplo: public void Ejecutar() { ... } private string ObtenerVersionEnsamblado(Assembly libreria) { ... } Para el caso específico de métodos en la capa de datos y la replicación en las capas superiores, siempre utilizar el siguiente formato: - Cuando se devuelve un objeto: [Entidad]Leer - Cuando se devuelva una lista: [Entidad]Listar - Para el CRUD se utilizará: DM[Entidad] - Las variaciones que puedan existir deben tomar la base del formato anterior: [Entidad]Leer/Listar[Variación] Ejemplo: public BEPersona PersonaLeer() { ... } public BEPersona PersonaLeerPorEstado() { ... } public List<BEPersona> PersonaListar() { ... } public List<BEPersona> PersonaListarPorEdad() { ... }

	public void DMPersona() { ... }
Propiedad	Utilizar estilo de escritura PascalCase. El nombre de la propiedad debe representar la entidad que devuelve. Nunca utilizar los prefijos "Get" o "Set". Ejemplo: public string Nombre { get; set; }
Campo (Público, Protegido o Interno)	Utilizar estilo de escritura PascalCase. Evitar el uso de campos no privados. Se debe usar propiedades en lugar de estos campos. Ejemplo: public string Nombre;
Campo (Privado)	Los campos privados siempre serán escritos utilizando como prefijo un "_", y debe tener el mismo nombre de la propiedad. Ejemplo: private string _Nombre;
Constante	Las constantes serán escritas en mayúsculas.
Campo Estático	Utilizar estilo de escritura PascalCase.

Enumerador	Utilizar estilo de escritura PascalCase.
	Ejemplo:
	public enum TiposCliente
	{
	Normal,
	Frecuente
	}

Delegado o Evento	Tratar como campo. Ejemplo: public event EventHandler CargarComponente;
Variable (Interna)	Utilizar estilo de Camel Pascal. Evitar el uso de nombres de un solo carácter tales como "x" o "y" a excepción de aquellos usados en la sentencia for. Evitar enumerar nombre de variables como cadena1, cadena2, cadena3, etc.
Parámetro	Utilizar estilo de escritura PascalCase. Ejemplo: public void Ejecutar(string Texto, int Iteraciones) { ... }

4.1.2. Estilo de Codificación

Las siguientes secciones describen el camino preferido para implementar código fuente en C#, para así crear un código legible, claro y consistente que sea fácil de comprender y mantener.

4.1.2.1. Formato

a. Nunca declarar más de un espacio de nombres por archivo.
b. Evitar colocar múltiples clases en un mismo archivo.
c. Siempre colocar llaves ("{" y "}") en líneas separadas.
d. Siempre usar llaves ("{" y "}") en sentencias condicionales.
e. Siempre usar Tab e indentación de tamaño 4.
f. Declarar cada variable independientemente – nunca en la misma sentencia.

Colocar la sentencia "using" o "Imports" en la parte superior del archivo. El grupo de los espacios de nombres de .NET colocarle por encima de los espacios de nombres particulares y todos en ordenados alfabéticamente. Por ejemplo:

```
using System;
using System.Collections
using System.Data;
using System.Data.SqlClient;
using System.Xml;
```

g. Agrupar la implementación interna de las clases por el tipo de miembros en el siguiente orden:
   - Campos
   - Propiedades
   - Constructores y Destructores
   - Métodos
   - Enumeradores, Estructuras o Clases internas

h. Ordenar las declaraciones a partir de los modificadores de acceso y visibilidad:

- Privados
- Internos
- Protegidos
- Público

i. Utilizar #region o #Region "" para agrupar los tipos de miembros de la clase.
j. Recursivamente indentar todos lo bloques de código en medio de llaves.
k. Evitar declarar múltiples atributos en una misma línea. En lugar de ello colocar cada atributo en una sentencia separada. Por ejemplo:

```
// Mal!
 [Atributo1, Atributo2, Atributo3]
public class MiClase
{ ... }

// Bien!
[Atributo1]
[Atributo2]
[Atributo3]
public class MiClase
{ ... }
```

4.1.2.2. Comentarios

a. Usar // o /// pero no /* ... */
b. No utilizar marcos de asteriscos.

Ejemplo:

```
//*****************************************
// Cuadro de comentarios
//*****************************************
```

c. Tratar de usar comentarios en el interior de los métodos para explicar aspectos globales del algoritmo.
d. No utilizar comentarios para explicar código obvio.
e. Incluir lista de tareas para utilizar los filtros de comentarios. Por ejemplo:

```
// TODO: Obtener cadena de conexión del registro
```

f. **SIEMPRE** aplicar bloques de comentarios XML propios de C# (///) a las declaraciones de los miembros públicos, protegidos e internos.

4.1.3. Uso del Lenguaje

4.1.3.1. General

a. Nunca omitir los modificadores de acceso. Declare explícitamente todos lo identificadores con su apropiado modificador de acceso en lugar de dejarlo con el de por defecto. Por ejemplo:

```
// Mal!
void EscribirEvento(string mensaje)
{...}

// Bien!
private void EscribirEvento(string mensaje)
{...}
```

4.1.3.2. Variables y Tipos

a. Tratar de inicializar las variables cuando las declare.

b. Siempre utilizar los tipos de dato de C# en lugar de los tipos del CTS de .NET. Por ejemplo:

```
short NO System.Int16
int NO System.Int32
long NO System.Int64
string NO System.String
```

c. Solo declarar a los campos como privados. Usar propiedades para acceder a los campos privados utilizando los modificadores de acceso de tipo público, protegido o interno.
d. Evitar el uso de números mágicos. En lugar de ello, usar constantes o enumeradores.
e. Evitar declarar variables de cadena con valores literales. En lugar use constantes, recursos, registros de sistema u otro tipo de repositorio de datos.
f. Solo declarar constantes para tipos de datos simples.
g. Declarar variables readonly o static readonly en lugar de constantes de tipos complejos.
h. Evitar utilizar conversiones directas (cast). En su lugar, utilizar el operador "as" y verificar por nulos (null). Por ejemplo:

```
object data = CargarData();
DataSet ds = dataObject as DataSet;

if (ds != null)
{...}
```

i. Siempre inicializar explícitamente los tipos referenciados en un arreglo al recorrer un bucle.

j. Evitar encapsular o desencapsular variables de tipo valor. Por ejemplo:

```
int fila = 1;

//Encapsulamiento implicito.
object objFila = fila;

//Desencapsulamiento explicito.
int nuevaFila = (int)objFila;
```

k. Tratar de usar el prefijo "@" en las cadenas literales en lugar de utilizar escapes.
l. Preferir usar StringBuilder en lugar de concatenación de cadenas.
m. No comparar cadenas a String.Empty o "" para verificar cadenas vacías. En su lugar, comparar por su longitud (String.Length) a cero.

4.1.3.3. Control de Flujos

a. Evitar invocar métodos en expresiones condicionales.
b. Evitar la creación de métodos recursivos. Utilizar bucle en su lugar.
c. Evitar usar foreach para iterar colecciones de objetos.
d. Utilizar el operador condicional ternario solo para condiciones triviales. Evitar operaciones complejas con este operador. Por ejemplo:

```
int resultado = esValido ? 9 : 4;
```

e. Evitar evaluar condiciones booleanas contra true o false. Por ejemplo

```
// Mal!
if (esValido == true)
{...}

// Bien!
if (esValido)
{...}
```

f. Evitar asignación de variables en expresiones condicionales. Porjemplo:

```
if ((i=2)==2)
{...}
```

g. Evitar expresiones condicionales complejas, usar variables booleanas para separar la expresión en expresiones manejables. Por ejemplo:

```
// Mal!
if ((valor > this._highScore) && (valor != this._highScore)) && (valor < this._maxScore))
{...}

// Bien!
esHighScore = (valor >= this._highScore);
esEmpate = (valor == this._highScore);
esValido = (valor < this._maxScore);

if ((esHighScore && !esEmpate) && esValido)
{...}
```

h. Solo usar expresiones switch/case para operaciones simples con condicionales lógicas.
i. Preferir if/else anidadas sobre switch/case para secuencias condicionales cortas y condiciones complejas.

4.1.3.4. Control de Excepciones

a. No usar los bloques try/catch para control de flujos.
b. Solo capturar las excepciones cuando van a ser controladas.
c. Nunca declarar un bloque match vacío.
d. Evitar anidar bloques try/catch en otro bloque catch.
e. Usar filtros de excepciones específicas.
f. Ordenar los filtros de excepciones de más especifica a más genérica.
g. Evitar el relanzamiento de excepciones.
h. Solo usar el bloque finally para liberar recursos utilizados en el bloque try.
i. Siempre utilizar validaciones para evitar excepciones.
j. Evitar definir excepciones personalizadas. Usar las clases de excepciones existentes.
k. Cuando una excepción personalizada es necesaria;
   - Siempre derivar de Exception no de ApplicationException
   - Siempre sobrescribir el método ToString().
   - Siempre sobrescribir el operador implicito string cuando se sobrescriba el método ToString().
   - Siempre implementar los siguientes patrones de constructores

```
[C#]
public MiExcepcion();
public MiExcepcion(string mensaje);
public MiExcepcion(string mensaje, Exception innerException);
```

l. Cuando se lancé una nueva excepción, siempre pasar la innerException para poder mantener la pila de excepciones.

4.1.3.5. Eventos, Delegados e Hilos

a. Siempre chequear que las instancias de los eventos y delegados sean diferentes de nulo.
b. Usar por defecto las clases EventHandler y EventArgs para los eventos más simples.
c. Siempre usar la palabra clave "lock" en lugar del tipo Monitor.
d. Solo bloquear objetos privados o estáticos privados.
e. Evitar bloquear los tipos de objetos. Por ejemplo:

```
lock(typeof(MiClase));
```

f. Evitar bloquear la instancia actual de un objeto. Por ejemplo:

```
lock(this);
```

4.1.3.6. Composición de Objeto

a. Siempre declarar los tipos explícitamente con un espacio de nombres. Nunca usar el espacio de nombres por defecto "{global}".
b. Evitar declarar métodos con más de 7 parámetros. Si fuera necesario considerar para una estructura o clase.
c. No usar la palabra reservada "new" para ocultar miembros de tipos derivados.
d. Solo usar la palabra reservada "base" o "MyBase" cuando se invoque en el constructor o implementación de una sobre escritura de algún método.
e. No usar el modificador de acceso protected con sealed en las clases.
f. Considerar usar sobrecarga en lugar de params como parámetro.
g. Siempre validar una variable de tipo enumerador antes de utilizarla dentro de un método. Esta variable puede contener cualquier valor que el tipo Enum puede soportar (por defecto es int). Por ejemplo:

```
public void Test(CategoriaLibros cat)
{
    if (Enum.IsDefined(typeof(CategoriaLibros), cat))
    {...}
}
```

h. Siempre llamar al método Close() o Dispose() en las clases que lo implementen.

4.1.3.7. Validaciones de texto en formularios:

Para el caso de las validaciones de texto ingresado por los formularios, deberá hacerse uso de funciones javascript las cuales deberán residir en un archivo de extensión ".js", dentro de una carpeta denominada "scripts" en el proyecto.

Las funciones que hagan validaciones de texto en background, deberán realizarse de forma asíncrona con acceso a datos mediante un Dataservice, los métodos contenidos serán precedidos por "`[OperationContract]`".

Ejemplo:

Función javascript.

```
function ValidaXYZ(x, y, z)
{
      var aValor = new Array(x, y, z);

      DataServices.XYZExiste(aValor, onSuccessValidaXYZ,
      onFailureValidaXYZ);
}
```

Método del Dataservice.

```
[OperationContract]
public bool XYZExiste(string[] aValor)
{
        bool bExiste = false;
        ...
        return bExiste;
}
```

5. Documentación de programas fuente

El objetivo de este acápite es el de establecer las pautas para que el equipo de programación realice una eficiente documentación de programas fuentes de la aplicación evitando saturarles con

agotadoras y extensas actividades de documentación a nivel de códigos, pero a la vez asegurando que los programas fuentes contengan en sí mismos la información suficiente que permita conocer la evolución.

5.1. Documentación en la creación de programas fuente nuevos

Para el código fuente que es creado:

5.1.1. En la cabecera del programa fuente

En la cabecera del programa fuente deben consignarse los siguientes datos de manera obligatoria:

- Código[1] y nombre del desarrollador.
- Fecha de creación.
- Si el programa implementa una o varias funcionalidades, deberá describirse las mismas en sólo una línea.
- Si el programa es generado por alguna herramienta de generación de código, deberá consignarse el nombre y versión de la herramienta.
- Opcionalmente, el número y fecha del documento que ordena la creación del programa.

5.1.2. En el cuerpo del programa fuente

En el cuerpo del programa se deberá documentar en los siguientes casos.

- Si se trata de un algoritmo complicado.
- Si se desea indicar algún aviso a los desarrolladores que en el futuro podrían dar mantenimiento al programa en curso, como por ejemplo la indicación de la corrección de algún código duro.

5.2. Documentación en el mantenimiento de programas fuente

Se considera un programa en mantenimiento a los programas que deben ser modificados por alguna corrección o mejora siempre que previamente hayan pasado a la etapa de producción o preproducción.

Al inicio del programa y como parte de los comentarios de cabecera se deberá indicar:

- El correlativo de mantenimiento[2].
- La fecha de inicio del proceso de mantenimiento.
- De existir, debe anotarse el indicador del documento que ha originado el mantenimiento.
- La descripción de la corrección que se va a realizar.

Una vez que se ha preparado el programa con los comentarios de la cabecera, se procede a realizar el cambio y a documentar el mismo:

- Se debe comentar el sector (o sectores) de código que quede en desuso, indicando al final del comentario el correspondiente correlativo de mantenimiento.
- El código fuente de reemplazo deberá ser colocado antes del código comentado delimitando el inicio y el fin del mismo con sendo comentarios que contengan el correlativo de mantenimiento con los prefijos *./(inicio)* y *./(final)* respectivamente.

MINISTERIO DE EDUCACION VISACION Oficina de Informática

MINISTERIO DE EDUCACION VISACION Oficina de Informática

MINISTERIO DE EDUCACION VISACION

[1] Si no se hubiera establecido el código de desarrollador, se deberá usar la inicial de su primer nombre unido al apellido paterno completo. En caso de homonimia a este nivel, se deberá usar como diferenciador la primera letra del apellido materno.
[2] Este correlativo está constituido por la cadena MANT/ donde / es el número correlativo que corresponde.

1591-2011-ED

CAPÍTULO III

6. Lineamientos para el tratamiento de la base de datos

6.1. Generalidades

El desarrollo de aplicaciones en la OFIN está íntimamente relacionado al proceso de modelamiento de la aplicación con una fuerte orientación a los *conceptos* en el entendido que los conceptos prevalecen y trascienden sobre los procesos. Es por ese motivo que el modelo conceptual se convierte en un artefacto de vital importancia en todo el proceso de desarrollo de la aplicación así como también en etapas posteriores como el mantenimiento y la correspondiente documentación.

Para la elaboración del Diseño de la Base de Datos se deben seguir las siguientes prácticas.

6.1.1. Herramienta de modelamiento

Es obligatoria la utilización de una herramienta de modelamiento en cada proyecto. Se deberá usar alguna herramienta que ofrezca entre sus características se cuente a la presencia de ingeniería reversa y generación de modelos en formato XMI.

Se deberá usar la siguiente secuencia de trabajo con la herramienta de modelamiento:

Modelo Conceptual –> Modelo Físico: Para proyectos que inician completamente.
Ingeniería Reversa –> Modelo Físico –> Modelo Conceptual. Para proyectos que se soportan en una base de datos previamente existente.

Todos los modelos de todas las aplicaciones residirán físicamente en un solo lugar y el acceso a los mismos es administrado por quien ocupa el rol del arquitecto de aplicaciones en la Oficina de Informática.

6.2. Construcción del Modelo Conceptual.

El modelo conceptual se debe construir bajo las siguientes consideraciones:

a. Debe existir una versión única del modelo conceptual de cada aplicación y su actualización es responsabilidad del jefe de proyecto o del especialista debidamente designado.

b. Diferenciación de entidades: Las entidades podrían usar la siguiente diferenciación de colores

- Azul: Entidades que son centrales, cuya actualización se debe realizar a nivel del ente gestor de la aplicación (o por actores centrales). Por ejemplo: Tipo de documento.
- Celeste: Entidades que son distribuidas o cuya actualización se debe realizar a nivel de los actores operacionales. Por ejemplo: Registro de notas.
- Naranja: Entidades externas a la aplicación, generalmente son propiedad de otra aplicación.
- Rojo: Son entidades principales de la aplicación.

c. Especificaciones de entidades:

- El nombre de la entidad debe ser en singular para los casos de entidades maestras. Se debe evitar los nombres de más de dos palabras y se deben evitar las preposiciones y artículos dentro de los nombres de las entidades, así como también las abreviaturas. Ejemplos de nombres de entidades

Correcto	Incorrecto	Observación
Tipo Documento	Tipo de Documento	Es incorrecto por tener más de dos palabras. Se ha suprimido la preposición.
Estudiante	estudiante	Es incorrecto por empezar con minúscula.
Contenido	Contenidos	Es incorrecto por estar en plural.
Institución Educativa	Inst_Educ	Es incorrecto por ser una abreviatura

- Las entidades de tipo colección que dependen de un padre se nombran de la siguiente manera:

  El nombre de la entidad que forma la colección (el hijo) va en plural, luego se coloca la palabra "por" y finalmente el nombre de la entidad padre. Por ejemplo:

Entidad padre	Entidad hijo	Entidad Relación
Institucion Educativa	Persona	Personas por Institución Educativa
Institucion Educativa	Año académico	Años por Institución Educativa

  Generalmente estas entidades resuelven las relaciones de muchos-a-muchos.

- Se deben priorizar las relaciones muchos a muchos sobre las relaciones uno a muchos. Esta práctica promoverá el desarrollo de sistemas de información que se adapten mejor a los cambios de los usuarios. No debe dejarse a la herramienta de modelamiento a que resuelva las relaciones de muchos-a-muchos.

d. Especificaciones de atributos:

- Se debe nombrar a los atributos con un nombre totalmente descriptivo. Por ejemplo:

Persona			
id persona	<pi>	Integer	<M>
apellido paterno		DESCRIPCION_CORTA	
apellido materno		DESCRIPCION_CORTA	
nombres		DESCRIPCION_CORTA	
domicilio		DESCRIPCION_LARGA	
sexo		ENUMERADO	
fecha nacimiento		FECHA	
tipo sangre		ENUMERADO	
con vida		ENUMERADO	
Identifier_1 <pi>			

- Los nombres de los atributos que forman parte de la clave primaria deberán iniciar con los caracteres "id" (Identificador).

- Se debe usar en lo posible un conjunto de dominios para uniformizar los tipos de dato de los atributos de las entidades. Algunos de los dominios utilizados a la fecha son:

Nombre del dominio	Tipo de dato
Abreviatura	Variable characters (10)
CantidadEntera	Integer
descripcion corta	Characters (30)
descripcion larga	Characters (80)
Enumerado	Short integer
Fecha	Date
Horaminuto	Characters (4)
Memo	Text

- El dominio Enumerado se usa para un conjunto finito de opciones. En la interfaz de usuario (desktop o web) se convierten en combos.

6.3. Generación del Modelo Físico.

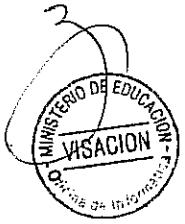
El modelo físico se generará siempre a partir del modelo conceptual dejando a la herramienta de modelamiento que determine los nombres, relaciones y demás objetos de la base de datos.

Deberá evitarse las modificaciones en el modelo físico de la base de datos dado que éstas deben ser realizadas en el modelo conceptual. Las únicas actualizaciones permitidas para el modelo físico deben ser las que no podrían realizarse sobre el modelo conceptual. Cuidar que en el modelador debe existir una manera de preservar las modificaciones en el modelo físico.

1591-2011-ED

En la generación de la base de datos a partir del modelo físico, deberán tenerse en cuenta los siguientes considerandos:

- Respetar los constraints de integridad referencial.
- Al generar la base de datos desde la herramienta de modelamiento debe cambiarse el tipo de letra a minúscula o mayúscula y el encoding correspondiente.
- Considerar que la base de datos del ambiente de desarrollo debe ser una base de datos homologada con la base de datos de producción. Todas las sentencias SQL que se elaboren en la aplicación deben respetar estrictamente la sensibilidad de mayúsculas y minúsculas presente en la base de datos.



1591-2011-ED

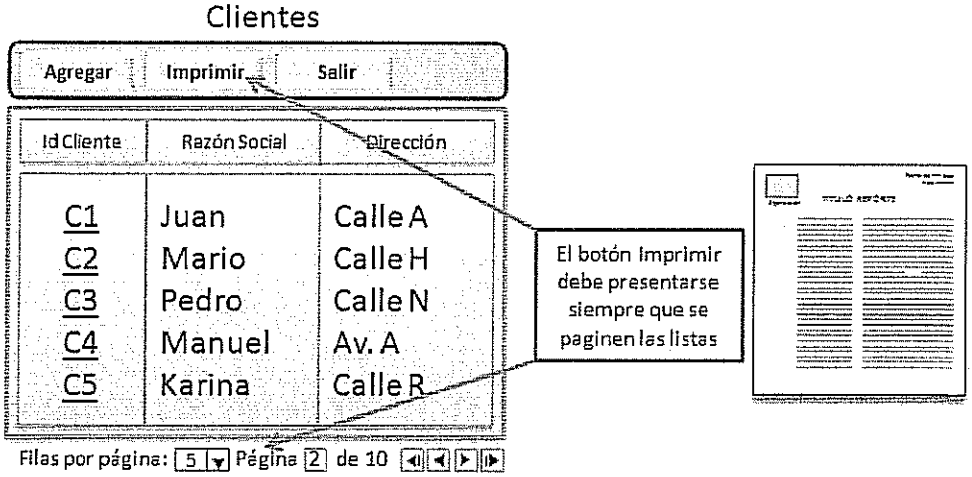
CAPÍTULO IV

7. Lineamientos para la interfase de usuario

7.1. Consideraciones generales para las interfases de usuario

Al momento de elaborar las interfases de usuario (a través de formularios, pop-pups, listas, etc.) se debe tener en cuenta las siguientes consideraciones generales:

- No deben existir errores ortográficos en las etiquetas, títulos, mensajes, advertencias, etc.
- Evitar las abreviaturas.
- Deben potenciar el uso del teclado y dejar en segundo plano el uso del mouse. Las aplicaciones deben estar preparadas para que se puedan realizar ingresos masivos de información y en la mayoría de las veces, los usuarios adquieren una gran pericia en el uso de los sistemas que es frecuente encontrar usuarios que sólo usan el teclado para sus actividades.
- Toda información que se muestre mediante páginas debe tener una opción de impresión que imprima toda la información. Esto debido a que el usuario no debería entrar a cada página para imprimir la información que necesita.



- Debe potenciarse la performance de las aplicaciones cuando se desarrolla la interfase de usuario, haciendo que los datos que se muestran en el cliente deben ser los obtenidos de la bases de datos del servidor o servidores. Por ejemplo, debe evitarse que se carguen combos con gran cantidad de datos de los cuales sólo se muestra uno solo o que en la paginación de listas, sólo deberá solicitar a la base de datos los registros mostrados en la página en curso.

- La interacción entre el usuario y la aplicación debe ser amigable y en todo momento los mensajes que se muestren deben ser correctamente entendidos por el usuario de tal manera que se minimice el tiempo de soporte de las aplicaciones.

- El 95% de las páginas deben medir menos de 70 Kb (medidos en el explorador). El 5% restante de las páginas de la aplicación no debe superar los 300 Kb.

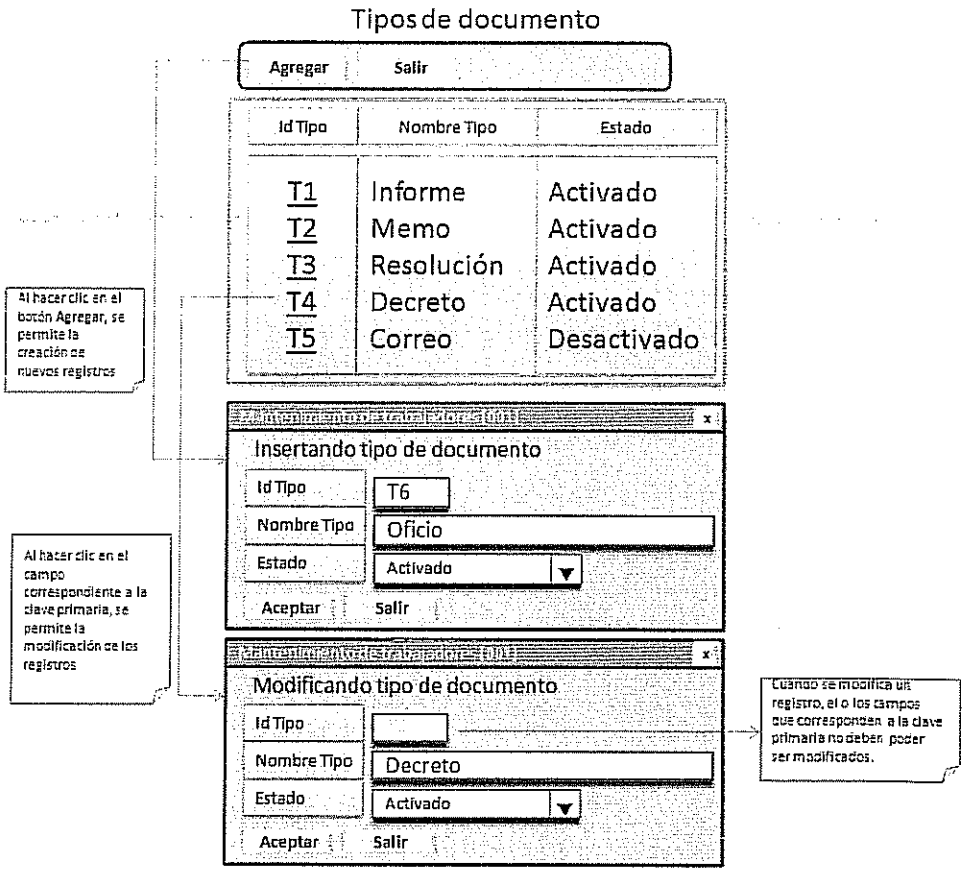
7.2. Especificaciones sobre las interfaces de usuario

a. Mantenimiento de tablas.

- Este tipo de mantenimiento muestra en primera instancia la lista de registros.
- Para agregar registros, deben acceder al botón Agregar, tras lo cual aparece el popup de Mantenimiento. El popup de inserción puede mantenerse activo para una siguiente inserción. Tras la inserción se debe actualizar la lista inicial.

1591 -2011-ED

- Para modificar un registro se accede al mismo a través del link establecido para el identificador del registro. Ver el gráfico:



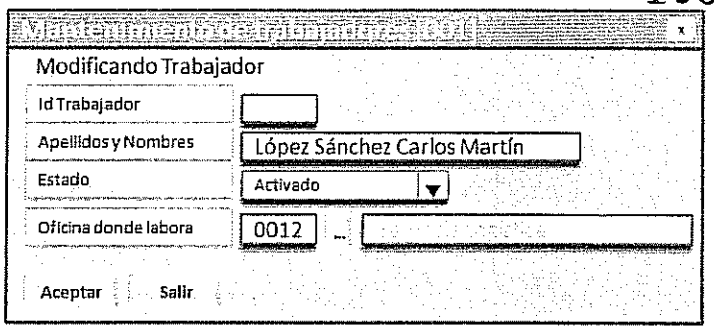
b. El popup de mantenimiento.

Los popups de mantenimiento son las ventanas que permiten el ingreso de datos al sistema. Las características principales de este mecanismo de ingreso a datos son las siguientes:

- Las validaciones son en línea, es decir, se valida el campo cuando se registra el dato (se debe usar Ajax).
- Siempre cuenta con un botón de Aceptar y un botón de Cancelar. Al aceptar se deben ejecutar todas las validaciones de campos que se hicieron al ir ingresando campo por campo.
- Se debe indicar en la parte superior de la ventana la operación que se está realizando sobre los campos del popup. Es decir, si se trata de una inserción o de una modificación se debe poner Insertando o Modificando respectivamente (seguido del nombre de la clase o entidad que se está implementando).
- En el titulo de la ventana siempre debe aparecer
  - el nombre de la ventana.
  - el código de la entidad a la que pertenece el usuario.
- Opcionalmente, los campos que son de llenado obligatorio se deben mostrar con un asterisco en la parte lateral izquierda preferentemente.

El gráfico siguiente muestra un ejemplo básico de cómo debe presentarse un popup:

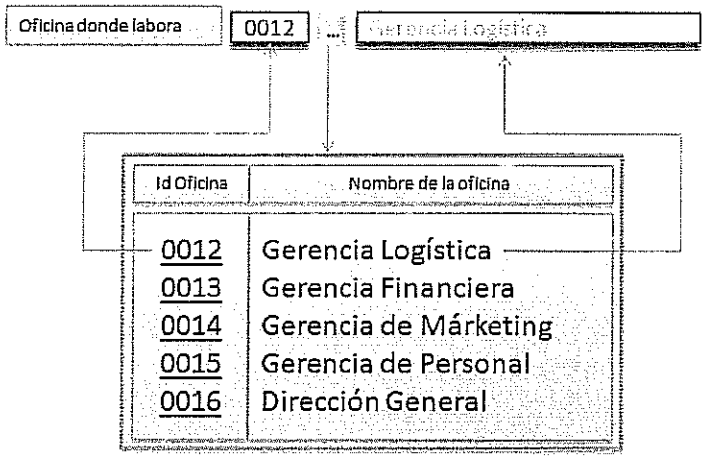
1591-2011-ED



c. Tratamiento de los diferentes tipos de datos

Los campos de texto:

- Deben validar siempre si permiten el registro de números o letras.
- Se debe impedir el ingreso de valores que excedan la longitud y valor numérico permitido por la base de datos.
- Los campos que hacen referencia a códigos de tablas foráneas deben ir acompañadas de un botón que muestre la lista de los registros disponibles. Adicionalmente debe acompañar un campo de texto (o etiqueta) que muestre la descripción correspondiente al código seleccionado o ingresado.



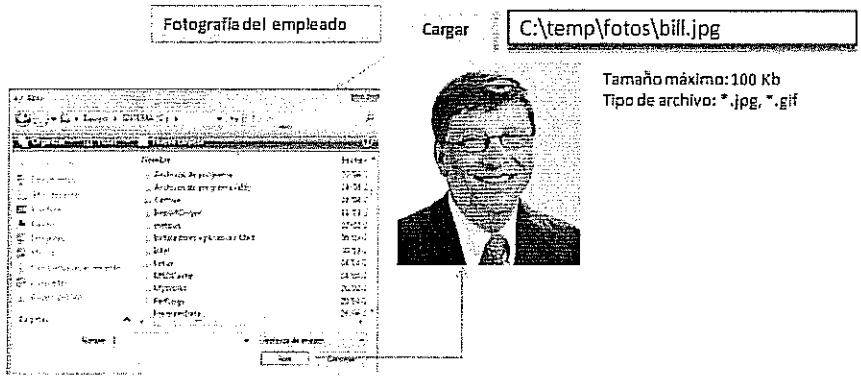
- Si el usuario no desea activar el botón de ayuda, por lo tanto debe conocer el código, entonces se debe usar la validación en background en forma asíncrona mediante el uso de javascript con acceso a datos mediante DataServices.

Los campos de fecha:

- Deberá ser siempre en el formato dd/mm/aaaa y debe validar que sea una fecha correcta.
- A menos que se especifique lo contrario deberá existir un calendario estándar de ayuda al ingreso de fechas en todas las aplicaciones. El uso del calendario no deberá impedir el registro del campo fecha directamente por teclado.

Imágenes y otros archivos:

- El campo imagen se actualiza mediante un botón de "Cargar" y la aplicación debe validar que la imagen no sobrepase un límite de Kbytes previamente establecido.
- También debe aparecer el formato de la imagen (gif y/o jpg) que se va a cargar.
- Los archivos de texto, de Word, Excel de la misma manera que las imágenes, deben tener una cantidad en Kbytes indicada y el formato posible.
- A menos que se indique lo contrario, todos los archivos se almacenarán en campos BLOB de las tablas correspondiente de la base de datos. Estas tablas deben contener SOLO las imágenes y el correspondiente identificador.

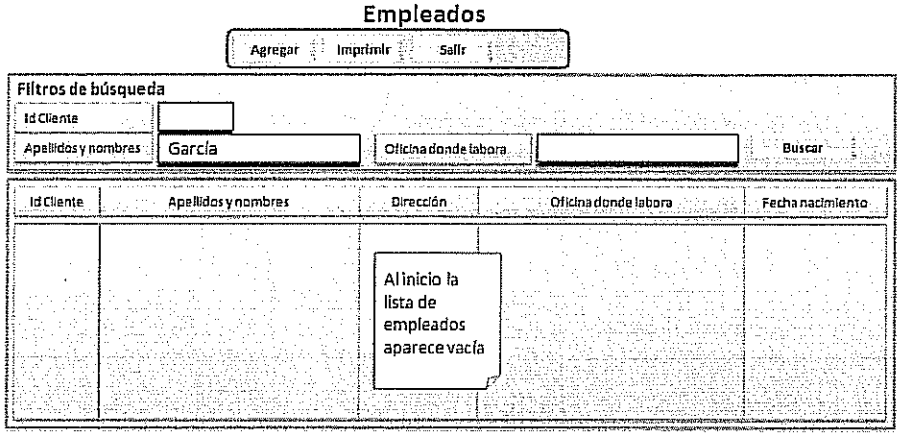


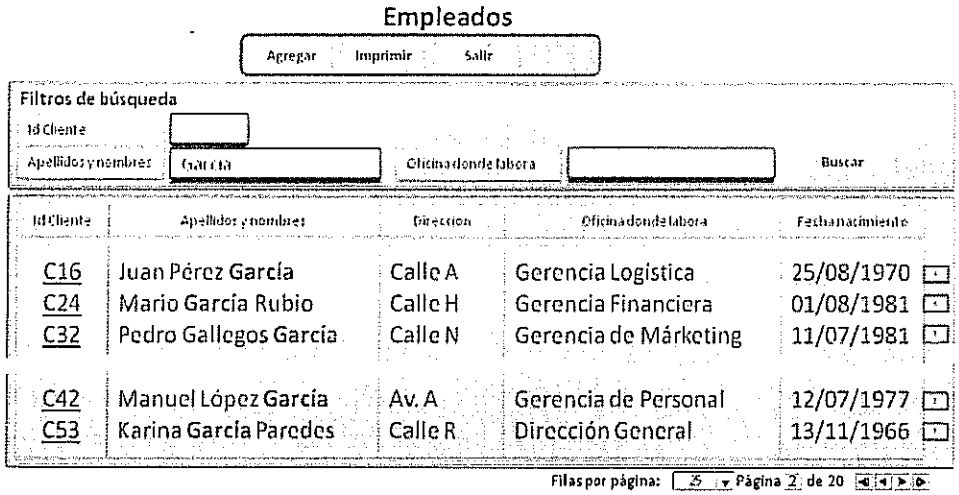
d. Tratamiento de listas

El tratamiento de listas es un aspecto que debe demandar mucho cuidado en la programación las aplicaciones. Para el tratamiento de listas se ha previsto las siguientes características generales:

Dependiendo del número esperado de registros:

- Si el número esperado de registros es grande entonces se debe mostrar la lista vacía en primera instancia y con las herramientas de filtro activadas.



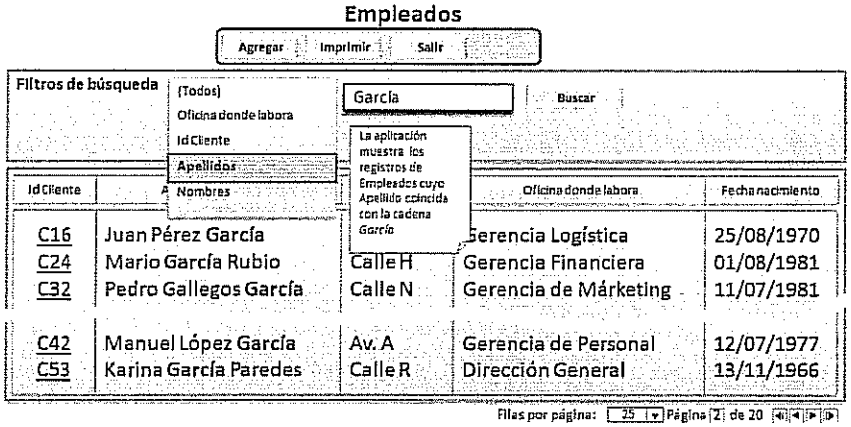


- Si el número esperado de registros es mediano, entonces se debe mostrar la lista paginada.
- En la eliminación de algún registro de la lista y ante las validaciones propias de la base de datos, los mensajes deben ser en un lenguaje entendible por el usuario.

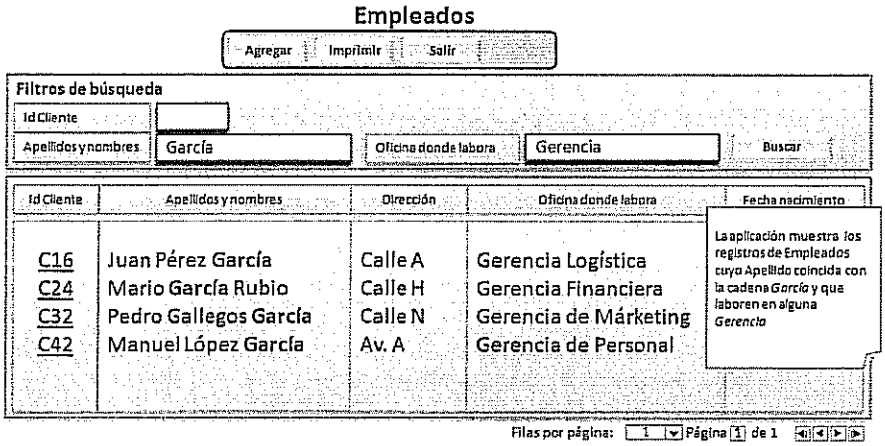
Los mecanismos de filtro:

Los mecanismos de filtros son opcionales y se justifican siempre y cuando la cantidad de registros de la Son:

- Tipo 1: Filtro por cualquier columna la lista.

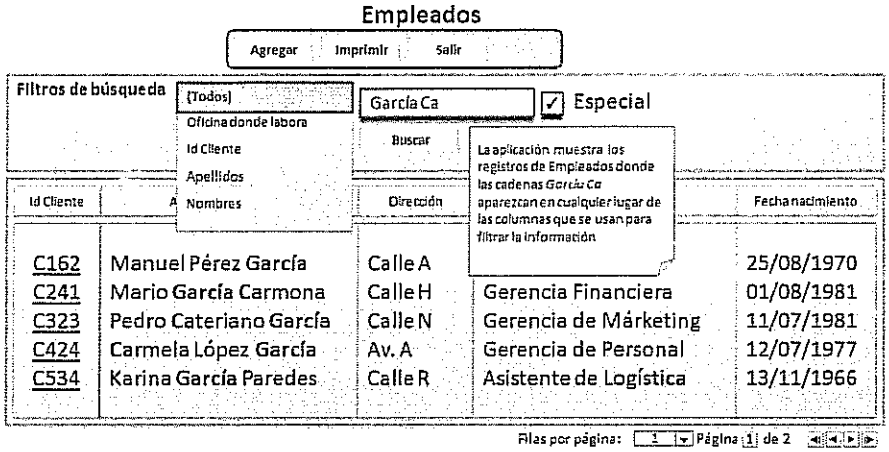


- Tipo 2: filtro por combinación de valores de columnas.



- Tipo 3: una variante del tipo 1 y aplica para todos los campos que se muestran en la lista. Consiste en un filtro por cadena pero, si en dicha cadena se aprecian dos o más palabras, se

debe hacer la búsqueda palabra por palabra. Este mecanismo de filtro deberá usarse discretamente siempre que se justifique, debido a que requiere un tiempo de respuesta mayor.



El ordenamiento de las columnas:

- Las listas deben tener un mecanismo de ordenamiento por columnas e indicar en todo momento mediante un símbolo el tipo de ordenamiento: si es ascendente o descendente.
- Todas las listas deben tener un mecanismo que ilumine la fila por donde se encuentre el puntero del mouse.
- Si el usuario selecciona un registro de la lista y tras ello la aplicación ejecute una acción adicional se deberá mostrar el registro en cuestión con el tipo de letra en negrita.

1591-2011-ED

CAPÍTULO V

8. Especificaciones de reportes

8.1. Concepto y la herramienta de elaboración de reportes

En adelante se denominará reporte a cualquier documento de salida de los sistemas de información que puedan ser elaboradas por la herramienta de reportes elegida[3] y que nuestra información relevante de la institución.

8.2. Tipos de reporte

De acuerdo a la forma de los reportes se considera la existencia de los siguientes tipos de reporte:

Listado: Se usa para mostrar predominantemente colecciones de forma tabular tienen generalmente cabeceras de tablas, por ejemplo.

Formulario: Este tipo de reporte imprime pares de atributo – valor se recomienda su uso cuando se desea mostrar información detallada de un registro en particular

Gerenciales: Son reportes que presentan generalmente gráficos (barras, pye, etc.) que difícilmente son susceptibles de formatear y con frecuencia mezclan componentes de tipo formulario o de listado.

Especiales: Son reportes muy elaborados y corresponden al formato previamente establecidos por una norma, directiva o estándar como requisitos.

8.3. Características de un reporte

El programador de reportes debe tener en cuenta que los reportes son componentes de software que están sujetas a los criterios de calidad.

a. El reporte debe ser fácil entendimiento para ello se debe:

- Usar preferentemente ANSI SQL en el reporte o acceder a procedimientos almacenados.
- Las sentencias SQL deben estar escritas con letras mayúsculas en las palabras reservadas. Tener el debido cuidado de la configuración de la BD si exige respetar el tipo de letra, esa especificación será relevante.

b. El reporte debe de promover un uso optimizado de los recursos de impresión: tóner, papel, impresora, teniendo en cuenta las siguientes consideraciones:
- Considerar evitar el uso de fondos oscuros (negro, azul, etc.) y letras blancas. En menor medida, evitar los tramados.
- Evitar imprimir poca información en el papel impreso, favoreciendo el desperdicio.
- Considerar que el reporte podría ser emitido por impresoras matriciales.

8.4. Lineamientos de forma para los reportes

Las siguientes son consideraciones para la forma de los reportes

Bordes del reporte: Los bordes deben ser de longitud de 1 cm para cada lado exceptuando el borde izquierdo que debe ser de 1.5 cm para que los agujeros del folder o encuadernación no impidan la visualización integral del reporte

Componentes del Reporte: Los reportes de tipo listado y de formularios deben de contar con los siguientes componentes

- **Logotipo institucional:** El logotipo debe ser ubicado en la parte superior izquierda del reporte y sus dimensiones deben ser de 1.5 cm x 1.5 cm

[3] La OFIN viene usando predominantemente la herramienta denominada Reporting Services para la elaboración de reportes.

1591-2011-ED

- **Título del reporte:** Debe estar ubicado a una distancia de 0.5 cm. del borde superior del reporte, con estas características.

Tipo de letra: Arial / Arial Narrow.
Tamaño de letra: Entre 12 y 14 puntos.
Resaltado: Si
Formato de letras: Todas en mayúsculas
Alineación: Centrado

- **Variables globales:** Van en la parte superior derecha y muestra información de la paginación, la fecha y hora del reporte Así:

Tipo de letra: Arial
Tamaño de letra: 6 ptos
Resaltado: No
Alineación: Derecha

- **Información general del reporte:** Esta información está constituida por los pares etiqueta – valor y deben tener las siguientes características:

Propiedad	Etiqueta	Valor
Tipo de letra	Arial	Arial
Tamaño	8 puntos	8 puntos
Negrita	Si	No
Formato	Todas en mayúsculas	Mayúsculas y minúsculas

El programador debe asegurarse que todos estos componentes aparezcan en todas las páginas del reporte.

Listas: Las listas corresponden a reportes de tipo *Listado*, y deben tener las siguientes características

Cabecera de la lista	Contenido de la lista
Tamaño de borde: 2 puntos Tamaño de letra: Arial o Arial Narrow Tamaño de letra: entre 4 y 8 puntos Resaltado: Si Alineación: Centrado	Tamaño de borde 1 punto Tipo de letra Arial Narrow Resaltado: No Tamaño de letra entre 4 y 8 puntos Alineación: Alfanuméricos alineados a la izquierda, cantidades alineadas a la derecha y valores enumerados centrado. Si la lista contiene imágenes , estas deberán tener una dimensión de 1 cm x 1 cm

MINISTERIO DE EDUCACION VISACION Oficina de Informática

MINISTERIO DE EDUCACION VISACION Oficina de Informática

MINISTERIO DE EDUCACION VISACION Oficina de Informática

CAPÍTULO VI

9. Funcionalidades de implementación obligatoria en las aplicaciones MED.

Todas las aplicaciones web distribuidas a ser desarrolladas tanto interna como externamente deben contener estas funcionalidades de manera obligatoria:

- La auditoría de tablas.
- La ayuda y mensajes de la aplicación.
- El acceso a la aplicación (log de accesos) y el control de acceso a través del el menú de acceso de la aplicación

9.1. La auditoría de tablas

Consiste en el registro permanente de todas las operaciones de actualización que se realicen sobre la base de datos de la aplicación.

Tiene por objetivo:

- Con el sistema en producción es posible hacer seguimiento en el manejo de la información de los usuarios en la aplicación. Si el usuario sabe que el registro de información puede ser auditado, entonces tenderá a hacer un registro de calidad.

- En la implementación de la aplicación, permitirá conocer detalladamente la evolución del uso de la aplicación.

Características

- Para los usuarios, esta la funcionalidad de auditoría debe ser totalmente transparente.
- Debe implementarse dentro del manejo de atomicidad de transacciones de la aplicación.
- Para cada actualización de tablas, se debe guardar la siguiente información:

  Fecha y hora de la operación.
  El usuario de la aplicación que generó la actualización.
  La tabla sobre la que se realizó la operación.
  El tipo de operación: Insert, Delete o Update.
  Si la operación fue INSERT, la lista de los pares campo-valor del registro que se insertó.
  Si la operación fue UPDATE, la lista de los pares campo-valor del registro antes de modificar y después de modificar.
  Si la operación fue DELETE, la lista de los pares campo-valor del registro que fue eliminado.

9.2. Los mensajes de ayuda de la aplicación

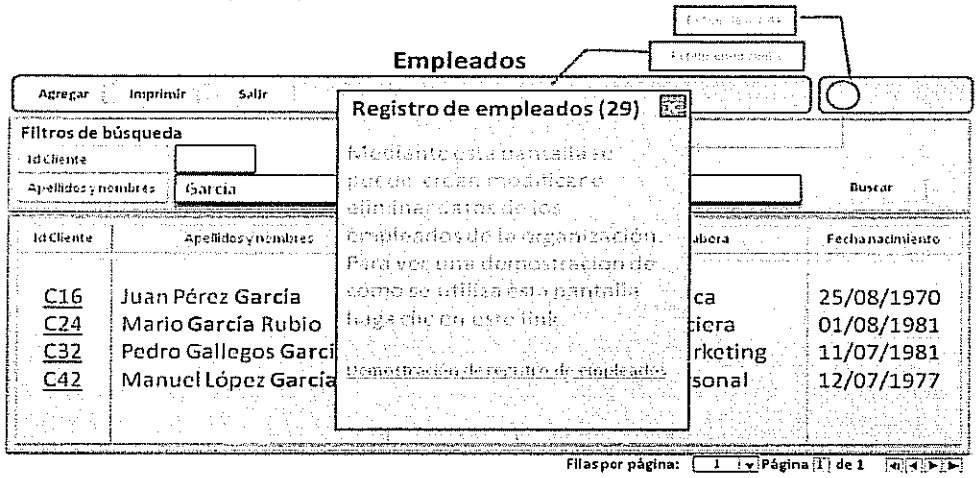
El uso de los mensajes de ayudas en las aplicaciones permitirá:

- Que los programadores se concentren en el desarrollo de la funcionalidad.
- Que los especialistas funcionales de la aplicación se responsabilicen del contenido del texto de ayuda.

Entre las características de esta funcionalidad se tiene:

- Es posible cambiar el texto y la forma de presentación de las ayudas y textos de las aplicaciones con la inclusión de mínimas modificaciones en los programas de la capa de presentación.
- Permite incluir información de tipo gráfica y se pueden hacer enlaces directos a fuentes de documentación externa a la aplicación
- Permite incluir código html para dar un formato especial al texto de ayuda.

- El mecanismo consiste en presentar un botón de ayuda que muestra la ayuda en una ventana emergente luego de que es presionado.



Para utilizar que esta funcionalidad pueda ser implementada en tiempo de desarrollo se procede de la siguiente manera:

El desarrollador deberá indicar en el código .ASP la etiqueta que corresponda a la ayuda que se desea mostrar consignando el número identificador de la ayuda para el sistema de información correspondiente. En el gráfico previo sería el número 29.

9.3. El acceso a la aplicación

Todos los sistemas de información distribuidos que se desarrollen en la institución deberán incluir o hacer interfase con el mismo mecanismo de control de acceso a las aplicaciones por parte de los usuarios, denominado Sistema de Administración Central.

El objetivo principal de esta funcionalidad es unificar los criterios para la administración de los permisos para ejecutar determinadas opciones a los usuarios.

Por otro lado, los usuarios de los diferentes sistemas podrán constar con un solo password a partir del cual podrá acceder a las diferentes aplicaciones siendo implementadas por la institución.

Permitirá controlar diferentes aspectos relacionados con el acceso, los que se enumeran a continuación.

- El ingreso por primera vez a las aplicaciones.
- La presentación del menú principal.
- La asignación de diferentes instituciones a un usuario determinado.
- El usuario podrá tener un menú de opciones diferenciado para cada institución a la que tiene acceso.