**ESTÁNDARES DE PROGRAMACIÓN**

**Business Consulting**

**Versión: v1.0**

**Historial de las revisiones**

| Ítem | Versión | Fecha | Autor | Descripción | Estado[[1]](#footnote-1) | Responsable (s) de Revisión y/o Aprobación |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1.0 | 29/12/2020 | LT | Versión Inicial | E |  |

**Autor:**

LT: Lizandro Torres

**CONTENIDO**

[1. FINALIDAD 8](#_Toc28241883)

[2. ALCANCE 8](#_Toc28241884)

[3. DEFINICIONES, SIGLAS Y REFERENCIA 8](#_Toc28241885)

[**3.1.** **DEFINICIONES** 8](#_Toc28241886)

[**3.2.** **SIGLAS** 10](#_Toc28241887)

[**3.3.** **REFERENCIA** 11](#_Toc28241888)

[4. AUDIENCIA 11](#_Toc28241889)

[5. METAS Y RESTRICCIONES ARQUITECTONICAS 11](#_Toc28241890)

[**5.1.** **METAS** 11](#_Toc28241891)

[**5.2.** **RESTRICCIONES** 11](#_Toc28241892)

[6. LINEAMIENTOS GENERALES 12](#_Toc28241893)

[**6.1.** **PARÁMETROS Y VARIABLES** 12](#_Toc28241894)

[**6.2.** **ARREGLOS** 12](#_Toc28241895)

[**6.3.** **ENUMERADOS** 12](#_Toc28241896)

[**6.4.** **MÉTODO** 13](#_Toc28241897)

[**6.5.** **CONSTANTE** 13](#_Toc28241898)

[**6.6.** **REGIONES** 13](#_Toc28241899)

[**6.7.** **DECLARACIONES DE CLASS E INTERFACES** 14](#_Toc28241900)

[**6.7.1.** **CLASE** 15](#_Toc28241901)

[**6.7.2.** **INTERFACE** 15](#_Toc28241902)

[**6.8.** **ESTRUCTURA DE CONTROL LÓGICA** 15](#_Toc28241903)

[**6.9.** **CONVENCIONES DE CÓDIGO** 18](#_Toc28241904)

[**6.9.1.** **DISEÑO** 18](#_Toc28241905)

[**6.9.2.** **INDENTACIÓN** 18](#_Toc28241906)

[**6.9.3.** **TAMAÑO DEL CÓDIGO FUENTE** 18](#_Toc28241907)

[**6.9.4.** **CANTIDAD POR LÍNEA** 18](#_Toc28241908)

[**6.9.5.** **INICIALIZACIÓN** 19](#_Toc28241909)

[**6.9.6.** **ESPACIOS EN BLANCO** 19](#_Toc28241910)

[**6.9.7.** **LÍNEAS EN BLANCO** 19](#_Toc28241911)

[**6.9.8.** **BÚSQUEDA DE INFORMACIÓN** 19](#_Toc28241912)

[**6.9.9.** **ROMPIENDO LÍNEAS** 20](#_Toc28241913)

[**6.9.10.** **CODIFICACIÓN** 21](#_Toc28241914)

[**6.9.11.** **COMENTARIOS** 21](#_Toc28241915)

[**6.9.12.** **NOMENCLATURA DE OBJETOS** 22](#_Toc28241916)

[**6.10.** **TIPOS DE PÁGINAS** 22](#_Toc28241917)

[**6.11.** **ESTRUCTURA DE UNA PÁGINA** 23](#_Toc28241918)

[**6.12.** **RECURSOS ESTÁTICOS** 24](#_Toc28241919)

[**6.13.** **VALIDACIÓN DE ENTRADA** 25](#_Toc28241920)

[**6.14.** **MENSAJE DE CONFIRMACIÓN, ÉXITO y ERROR** 26](#_Toc28241921)

[**6.15.** **REPORTE** 26](#_Toc28241922)

[**6.16.** **PAGINACIÓN** 26](#_Toc28241923)

[**6.16.1.** **EJECUCIÓN DE CONSULTAS** 27](#_Toc28241924)

[**6.16.2.** **LLAMADA A PROCEDIMIENTOS ALMACENADOS** 27](#_Toc28241925)

[**7.** **SEGURIDAD** 27](#_Toc28241926)

[**7.1.** **TRATAMIENTO DE EXCEPCIONES** 27](#_Toc28241927)

[**7.2.** **VALIDACIÓN DE DATOS** 28](#_Toc28241928)

[**8.** **PRUEBAS** 28](#_Toc28241929)

[**8.1.** **TIPOS DE PRUEBAS** 28](#_Toc28241930)

[**8.1.1.** **PRUEBA UNITARIA** 28](#_Toc28241931)

[**8.1.2.** **PRUEBA DE INTEGRACIÓN** 29](#_Toc28241932)

[**8.1.3.** **PRUEBA DE ACEPTACIÓN** 29](#_Toc28241933)

[**8.1.4.** **PRUEBA DE SERVICIOS WEB** 29](#_Toc28241934)

[**8.1.5.** **PRUEBA DE CARGA Y STRESS** 30](#_Toc28241935)

[**8.1.6.** **ANÁLISIS DE CÓDIGO ESTÁTICO** 30](#_Toc28241936)

[**8.2.** **PLAN DE PRUEBAS** 30](#_Toc28241937)

[**9.** **ENTREGA** 30](#_Toc28241938)

[**9.1.** **TIPOS DE ARCHIVOS QUE SE UTILIZAN EN UNA APLICACIÓN WEB** 31](#_Toc28241939)

# **FINALIDAD**

Ser una guía práctica para iniciar un nuevo proyecto de software de Business Consulting, a partir del establecimiento de un conjunto mínimo de requerimientos y recomendaciones técnicas que estandaricen el desarrollo de software con una serie de instrucciones de trabajo coherentes y un marco de referencia de terminología y vocabulario común.

# **ALCANCE**

El presente estándar es aplicable a la construcción de una nueva aplicación de software en entorno web que utilice la tecnología base indicada en este documento.

# **DEFINICIONES, SIGLAS Y REFERENCIA**

## **DEFINICIONES**

|  |  |
| --- | --- |
| **VOCABLO** | **DEFINICIÓN** |
| **Microsoft Visual Studio** | [Entorno](http://es.wikipedia.org/wiki/Entorno) integrado de desarrollo de software para sistemas operativos [Windows](http://es.wikipedia.org/wiki/Windows). Soporta múltiples lenguajes de programación tales como [C++](http://es.wikipedia.org/wiki/Visual_C%2B%2B), [C#](http://es.wikipedia.org/wiki/C_sharp), [Visual Basic .NET](http://es.wikipedia.org/wiki/Visual_Basic_.NET), [F#](http://es.wikipedia.org/wiki/F_Sharp), [Java](http://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)), [Python](http://es.wikipedia.org/wiki/Python), [Ruby](http://es.wikipedia.org/wiki/Ruby), [PHP](http://es.wikipedia.org/wiki/PHP); al igual que entornos de desarrollo web como [ASP.NET](http://es.wikipedia.org/wiki/ASP.NET) Web API, ASP.NET MVC. |
| **Framework** | Entorno de trabajo es una estructura conceptual y tecnológica de asistencia definida, normalmente, con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto. |
| **Dominio** | Concepto para diseñar e implementar aplicaciones empresariales complejas donde es fundamental definir un Modelo de Dominio expresado en el propio lenguaje de los expertos que dominan el negocio llamado Lenguaje Ubicuo (lenguaje común entre los programadores y los usuarios). |
| **IOC (Autofac)** | Patrón de diseño orientado a objetos que permite el desacoplamiento físico de las capas. Autofac es una librería open source que implementa dicho patrón y es la más empleada por su simplicidad de uso y performance. |
| **ORM (Automapper)** | Librería open source para el mapeo entre entidades de datos, de mucha utilidad para el intercambio de información entre las capas y niveles del software. |
| **ASP.NET Web API** | Framework que facilita la creación de servicios HTTP disponibles para una amplia variedad de clientes, entre los que se incluyen exploradores y dispositivos móviles. ASP.NET Web API es la plataforma perfecta para crear aplicaciones REST en .NET Framework |
| **Entity Framework (EF)** | Conjunto de tecnologías de ADO.NET que permiten el desarrollo de aplicaciones de software orientadas a datos. Objeto-relacional que permite a los desarrolladores de . NET trabajar con datos relacionales usando objetos específicos del dominio. Elimina la necesidad de la mayor parte del código de acceso a datos que los desarrolladores suelen tener que escribir. |
| **Patrón Unit of Work (UoW) o Unidad de Trabajo** | Este patrón tiene como objetivo tratar como una Unidad todos aquellos objetos nuevos, modificados o eliminados con respecto de una fuente de datos. Este patrón UoW resulta muy útil a la hora de persistir un conjunto de acciones a ejecutar sobre la base de datos, evitando el exceso de conexiones contra la misma. |
| **Patrón Repositorio (Repository Pattern)** | Un repositorio es un mediador entre el dominio de la aplicación y los datos que le dan persistencia. Con este planteamiento podemos pensar que el usuario de este repositorio no necesitaría conocer la tecnología utilizada para acceder a los datos, sino que le bastaría con saber las operaciones que nos facilita este “mediador”, el repositorio. |
| **Plantillas de texto T4** | En Visual Studio, una plantilla de texto T4 es una combinación de bloques de texto y lógica de control que puede generar un archivo de texto. La lógica de control se escribe como fragmentos de código.  Normalmente se utilizará varias plantillas que leen los datos en un archivo de entrada único o base de datos, y generaría algunos archivos con extensión cs, vb u otros archivos de código fuente. Cada plantilla genera un archivo. Se ejecutan dentro de Visual Studio en tiempo de diseño. |
| **Angular** | Angular es un framework TypeScript y/o JavaScript opensource de **desarrollo de aplicaciones web en el lado cliente**, creado y soportado por la comunidad de desarrolladores de Google y utiliza el patrón MVC (Model-View-Controller).  Este framework permite [**extender HTML** con tags personalizados](http://docs.angularjs.org/guide/directive), [definir y vincular (data-binding) variables vista/controllador](http://docs.angularjs.org/guide/dev_guide.templates.databinding), consultas ajax con peticiones HTTP, sistema óptimo de [templating](http://docs.angularjs.org/guide/dev_guide.templates" \t "_blank), manipulación de datos en JSON, [inyección de dependencias](http://docs.angularjs.org/guide/di), deep linking, [formularios de validación](http://docs.angularjs.org/guide/forms), desacoplamiento del DOM de Javascript. |
| **Knockoutjs** | Knockout (KO) es un framework JavaScript que ayuda a crear interfaces de usuario de visualización y edición ricas y con capacidad de respuesta con un modelo de datos subyacente limpio.  Este framework permite extender HTML con tags personalizados, definir y vincular (data-binding) variables vista/controllador, consultas ajax con peticiones HTTP, sistema óptimo de templating, manipulación de datos en JSON, inyección de dependencias. |
| **XAMARIN** | Xamarin es una plataforma para desarrollar aplicaciones para plataformas IOS y Android. |
| **Pascal Case** | El primer carácter de todas las palabras se escribe en Mayúsculas y los otros caracteres en minúsculas, por ejemplo: ApellidoPaterno. |
| **Camel Case** | El primer carácter de todas las palabras, excepto de la primera palabra se escribe en Mayúsculas y los otros caracteres en minúsculas, por ejemplo: apellidoPaterno. |

## **SIGLAS**

|  |  |
| --- | --- |
| **SIGLA** | **SIGNIFICADO** |
| **IE** | Internet Explorer |
| **SOAP** | Simple Object Access Protocol |
| **REST** | Representational State Transfer |
| **DDD** | Domain Driven Design |
| **HTTP** | HyperText Markup Language |
| **CSS** | Cascading Style Sheet |
| **JS** | JavaScript |
| **JSON** | JavaScript Object Notation |
| **WCF** | Windows Communication Foundation |
| **ORM** | Object-Relational Mapping |
| **MVC** | Design Pattern Model-View-Controller |

## 

## **REFERENCIA**

|  |  |
| --- | --- |
| **Referencia** | **Documento** |
| **Microsoft Ibérica** | Guía de Arquitectura N-Capas Orientada al Dominio con .NET 4.x |
|  |  |

# **AUDIENCIA**

La audiencia objetivo del presente documento corresponde a:

* El equipo de arquitectura de software; el cual utilizará este documento para la revisión de los artefactos de desarrollo, velando que se sigan los estándares aquí establecidos.
* Los miembros del equipo desarrollo, los cuales encontrarán en este documento el detalle de la documentación de los bloques básicos, en los que se construirá gran parte de los elementos de las aplicaciones.
* Los miembros del equipo calidad; Encontrarán la información necesaria para crear y aplicar listas de revisión e inspección.
* Los miembros del equipo mantenimiento, los cuales encontrarán en este documento el lineamiento base para la construcción de los componentes de software a ser mantenidos.

# **METAS Y RESTRICCIONES ARQUITECTONICAS**

## **METAS**

* El software permitirá el acceso a los usuarios desde cualquier PC de escritorio con un navegador con conexión a internet o a través de dispositivos móviles si aplica el requerimiento.
* El software debe atender las necesidades de concurrencia de acceso y carga, con un esquema apoyado en granjas de servidores (si el escenario lo amerita).
* La interfaz de usuario debe hacer uso de tecnologías de lado de cliente para el nivel de presentación (web 2.0) para aplicaciones web e interfaces nativas (formularios) para dispositivos móviles.
* La interfaz de usuario debe ser adaptativa (web responsive) y debe poderse acomodar a distintos dispositivos.
* Los componentes de cualquier desarrollo de software interno o realizado por terceros, deben estar alineado al Estándar de Arquitectura de Software vigente en la OTI.

## **RESTRICCIONES**

* El sistema deberá utilizar el SQL Server 2016 o superior como motor de base de datos.
* El sistema deberá utilizar Reporting Services 2016 Express o superior como generador de reportes.
* El sistema deberá ser desarrollado utilizando .NET Framework 4.5 o .Net Core: Microsoft AspNetCore v2.2 o superiores;
* El sistema deberá soportar los navegadores IE 10 o superior, y las últimas versiones de Mozilla Firefox 39 o superior y Google Chrome 47 o superior, Microsoft Edge.

# **LINEAMIENTOS GENERALES**

* Elegir nombres comprensibles y relacionados a la tarea que corresponda al objeto nombrado.
* Utilice lo menos posible las abreviaturas, ya que pueden causar confusiones, salvo para términos largos o utilizados con frecuencia.

## **PARÁMETROS Y VARIABLES**

* El nombre debe estar compuesto por una o más palabras en sustantivo, en notación Camel Case.
* Debe nombrarse en singular (excepto variables que almacenen un conjunto de elementos).
* Se debe evitar nombrar usando un sólo carácter, excepto variables temporales usadas en bucles como pueden ser i, j, k, m, y n.
* Un parámetro de un método debe ser instancia de un objeto.
* No use guiones bajos para nombres de variables locales.
* No use palabras reservadas para nombrar variables.
* Los nombres de las variables booleanas deben ser lo suficientemente claros como para expresar el valor que encierran ante una pregunta. Por Ejemplo: bool esActivo.
* Usar los tipos específicos de C#, en vez de los tipos definidos en el espacio de nombre System.

Por ejemplo:

int edad; (no Int16)

string nombre; (no String)

object contrato; (no Object)

## **ARREGLOS**

* El nombre debe estar compuesto por una o más palabras en sustantivo, en notación Camel Case.
* Utilice sintaxis concisa para inicializar las matrices en la línea de declaración.
* Usar preferentemente la instrucción foreach que ofrece una manera sencilla y limpia de iterar los elementos de una matriz.

## **ENUMERADOS**

* Para los enumerados y los valores de enumeración, no se tendrá que anteponer ningún prefijo, se usará la notación Pascal Case.
* Se utilizará sustantivos en singular para nombrar los enumerados.
* No usar prefijos ni abreviaciones.
* No usar el sufijo “Enum”, en tipos de enumeración.

Por Ejemplo:

public enum Estado

{

Inactivo = 0,

Activo = 1

}

## **MÉTODO**

* Para el nombre del método se debe utilizar la convención Pascal Case,
* El nombre debe estar compuesto por una o más palabras en verbo infinitivo.
* El nombre debe decir lo que hace el método.
* Evitar escribir métodos muy largos. Si un método tiene más de 25 líneas de código, debe considerar refactorizarlo en métodos separados.
* Si tienes un método que retorna una colección, devuelve una colección vacía en vez de null, si no hay datos que retornar.

Por Ejemplo:

public void ProcesarPlanilla()

{

//...

}

## **CONSTANTE**

* Para el nombre de las constantes se debe utilizar la convención Pascal, el nombre puede estar compuesto por una o más palabras, evita usar sólo mayúsculas.
* Declarar los valores como constante solo si se está seguro que este valor no cambiará.

Por Ejemplo:

public const string TipoCliente = "VIP";

## **REGIONES**

Se debe usar regiones para separar el código y tenerlo segmentado según su tipo; estas secciones deben ser las siguientes:

* **Variables:** Debe englobar todas las variables definidas.
* **Delegados y Eventos:** Si existe delegados y/o eventos se englobarán dentro de la misma región
* **Propiedades:** Sobre las propiedades que existan.
* **Constructores:** Si se han definido constructores personalizados.
* **Métodos:** Se englobarán todos los métodos.

Si algunos de los elementos no se encuentran definido, se debe eliminar la región correspondiente.

## **DECLARACIONES DE CLASS E INTERFACES**

* Ningún espacio en blanco entre el nombre de un método y el paréntesis "(" que abre su lista de parámetros
* La llave de apertura "{" aparece al final de la misma línea de la sentencia declaración
* La llave de cierre "}" empieza una nueva línea indentada para ajustarse a su sentencia de apertura correspondiente, excepto cuando no existen sentencias entre ambas, que debe aparecer inmediatamente después de la de apertura "{"

*class Cliente extends Persona {*

*int nombre;*

*string apellidoPaterno;*

*Cliente(int nombreCliente, string apellidoPaternoCliente) {*

*nombre = nombreCliente;*

*apellidoPaterno = apellidoPaternoCliente;*

*}*

*int metodoVacio() {}*

*...*

*}*

* Los métodos se separan con una línea en blanco
* Toda declaración de clase de un proyecto deberá declarar el namespace de donde pertenece; además, este corresponderá a la ruta física que se encuentra ubicado el objeto.

namespace Persistencia\_Licpro.Sdk.Contracts.Responses{

public class Notificacion {

…

}

}

## **CLASE**

* Debe estar compuesta por una o más palabras en sustantivo en singular.
* Usar el estilo Pascal Case, con la primera letra de cada palabra en mayúscula.
* En caso que la clase haga referencia a un objeto o entidad en la base de datos esta debe llamarse como la entidad que representa. Por ejemplo, si en la base de datos existe una tabla IOTDTD\_OFICINA la cual representa a la entidad Oficina el nombre de la clase seria Oficina.

## **INTERFACE**

* Debe estar compuesta por una o más palabras en sustantivo en singular.
* Usar el estilo Pascal Case, con la primera letra de cada palabra en mayúscula.
* La clase de tipo interfaz lleva el prefijo I.
* La clase que implementa una interfaz no lleva ni prefijo ni subfijo. Por ejemplo:

Interface: INotificacionRepository

Implementación de Interface: NotificacionRepository

## **ESTRUCTURA DE CONTROL LÓGICA**

Se debe utilizar las estructuras if, ifElse, switch, for/foreach, while/do-while para entender la secuencia de ejecución del bloque de código. Los bloques de código deben de estar englobados entre llaves y seguirán las siguientes reglas:

* Para la sentencia if, if-else, if else-if else; se debe evitar usar una sentencia simple de una sola línea, esto para una mejor lectura del código, por ello deben tener la siguiente apariencia:

if (condición) {

// acciones

}

if (condición) {

// acciones

} else {

// otras acciones

}

if (condición) {

// acciones

} else if (condición) {

// otras acciones

} else {

// otras acciones más

}

**Nota:**

* En general es una buena idea usar paréntesis en expresiones que implican distintos operadores para evitar problemas con el orden de precedencia de los operadores. Incluso sí parece claro el orden de precedencia de los operadores, podría no ser así para otros, no se debe asumir que otros programadores conozcan el orden de precedencia.

if (a == b && c == d) // EVITAR!

if ((a == b) && (c == d)) // CORRECTO

* La sentencia switch; debe tener la siguiente apariencia:

switch (condición) {

case A:

// acciones

Break;

case B:

// acciones

Break;

default:

// acciones

Break;

}

* En la sentencia for/foreach; se debe utilizar las llaves incluso cuando haya una sola sentencia en el bucle, por ello deben tener la siguiente apariencia:
  + Sentencia **for** debe tener la siguiente forma:

for ( int i = 0; i < 10; ++i ){

//acciones

}

* + Sentencia **foreach** debe tener la siguiente forma:

foreach ( int i in listaEnteros ){

// acciones

}

**Notas:**

* Para anexar cadenas en bucles, utilizar el objeto StringBuilder y no string, dado que permite una mejor performance del código.
* Siempre declare las variables en el bucle.
* No usar variables en decimales para definir los bucles.
* Para recorrer arreglos utilizar **foreach.**
* En la sentencia while/do-while; se deben tener la siguiente apariencia:
  + Sentencia **while** debe tener la siguiente forma:

while ( condición ){

//acciones

}

* + Sentencia **do while** debe tener la siguiente forma:

Do {

// acciones

} while ( condición );

**Nota:** Para anexar cadenas en bucles, utilizar el objeto StringBuilder y no string, dado que permite una mejor performance del código.

* En la sentencia try-catch; se deben tener la siguiente apariencia:

Try {

//acciones

} catch ( Exception ) {}

Try {

//acciones

} catch ( Exception e ) {

// otras acciones

}

Try {

//acciones

} catch ( Exception e ) {

// otras acciones

} finally {

// otras acciones finales

}

## **CONVENCIONES DE CÓDIGO**

## **DISEÑO**

* Utilizar una sola instrucción por línea.
* Utilizar una sola declaración por línea.
* Agregue al menos una línea en blanco entre las definiciones de método y las definiciones de propiedad.
* Usar una línea en blanco para separar un grupo lógico de código.

## **INDENTACIÓN**

* Usa TAB para la sangría. No usar ESPACIOS. Define el tamaño del Tab de 4 espacios.
* Los comentarios deben estar al mismo nivel que el código.

Por ejemplo:

//Formatea un mensaje

string mensajeCompleto = “Formato Final”;

## **TAMAÑO DEL CÓDIGO FUENTE**

* Se debe evitar el código repetido utilizando la refactorización.
* Se debe evitar la codificación de excesiva funcionalidad dentro de un mismo método, clase o página web.
* Deben evitarse los archivos de gran tamaño que contengan más de 2000 líneas.
* Evitar las líneas de más de 80 caracteres, ya que no son manejadas bien por muchas herramientas.

## **CANTIDAD POR LÍNEA**

Se recomienda una declaración por línea, ya que facilita los comentarios. En otras palabras, se prefiere

int nivel; // nivel de indentación

int tam; // tamaño de la tabla

antes que

int level, size;

No poner diferentes tipos en la misma línea. Ejemplo:

int foo, fooarray[]; //ERROR!

## **INICIALIZACIÓN**

Intentar inicializar las variables locales donde se declaran. La única razón para no inicializar una variable donde se declara es si el valor inicial depende de algunos cálculos que deben ocurrir.

## **ESPACIOS EN BLANCO**

Se utilizará un espacio en blanco en los siguientes casos:

* Separar una palabra clave y su paréntesis asociado. Se aplica a las siguientes palabras clave: catch, for, if, switch, while.
* Separar una palabra clave que lleva un argumento, por ejemplo: return true;
* Entre dos palabras clave.
* Entre una palabra clave o un paréntesis de cierre y una apertura de llaves “{“.
* Después de una coma en una lista.
* Después de un punto y coma en una sentencia for: for (expr1; expr2; expr3) {.

No se utilizará un espacio en blanco en las siguientes situaciones:

* Entre el nombre de un método y la apertura de paréntesis

## **LÍNEAS EN BLANCO**

Se utilizarán líneas en blanco en los siguientes casos:

* Para separar los comentarios de cabecera del fichero.
* Entre declaraciones de clases.
* Entre declaraciones de métodos.
* En una clase, entre la última variable y el primer método.
* Para agrupar secciones de código que estén lógicamente relacionadas.

## **BÚSQUEDA DE INFORMACIÓN**

* Se debe ejecutar una búsqueda de información sin discriminar mayúsculas, minúsculas o acentuación.

## **ROMPIENDO LÍNEAS**

Cuando una expresión no entre en una línea, romperla de acuerdo con estos principios:

* Romper después de una coma.
* Romper antes de un operador.
* Preferir roturas de alto nivel (más a la derecha que el "padre") que de bajo nivel (más a la izquierda que el "padre").
* Alinear la nueva línea con el comienzo de la expresión al mismo nivel de la línea anterior.
* Si las reglas anteriores llevan a código confuso o a código que se aglutina en el margen derecho, indentar justo 2 TAB (8 espacios) en su lugar.
* Saltar de líneas por sentencias if deberá seguir generalmente la regla de los 2 TAB (8 espacios), ya que la indentación convencional 1 TAB (4 espacios) hace difícil ver el cuerpo. Por ejemplo:

*//NO USAR ESTA INDENTACION*

*if ((condicion1 && condicion2)*

*|| (condicion3 && condicion4)*

*||!(condicion5 && condicion6)) { //MALOS SALTOS*

*hacerAlgo(); //HACEN ESTA LINEA FACIL*

*DE OLVIDAR*

*}*

*//USE ESTA INDENTACION*

*if ((condicion1 && condicion2)*

*|| (condicion3 && condicion4)*

*||!(condicion5 && condicion6)) {*

*hacerAlgo();*

*}*

## **CODIFICACIÓN**

* Se debe adoptar el método de codificación UTF-8 para el código fuente en lugar de ISO-8859-1 (por defecto en MS Windows, no así en Linux o Mac OS), para evitar problemas posteriores a la hora de generar documentación desde el código fuente.
* Se debe configurar el editor de texto IDE utilizado para que almacene el código fuente en UTF-8 (Sólo es necesaria en Windows porque Linux trabaja por defecto con UTF-8).

## **COMENTARIOS**

Estos comentarios se utilizan para describir el código ("el cómo"), y en ellos se incluye información relacionada con la implementación, tales como descripción de la función de variables locales, fases lógicas de ejecución de un método, captura de excepciones, etc.

* No utilizar comentarios al final de una instrucción con excepción a las declaraciones.
* Colocar el comentario en la línea superior del código que corresponda al comentario.
* Comenzar el texto del comentario con una letra mayúscula.
* Finalizar el comentario con un punto.
* Insertar un espacio entre el delimitador de comentario ( //) y el comentario.
* Se debe emplear comentarios en los siguientes casos:
  + Si se declara alguna clase, método, variable, etc., sin ponerle un nombre explícito.
  + Si se usa una lógica compleja.
  + Si se inicializa una variable en un valor distinto a 0, -1, etc.
  + En caso que el desarrollador estime que sea necesario.
* Los comentarios deben escribirse con una gramática y puntuación adecuada.
* Distinguimos tres tipos de comentarios de implementación:
  + **COMENTARIOS DE BLOQUE:**

Permiten la descripción de ficheros, clases, bloques, estructuras de datos y algoritmos.

*/\**

*\* Esto es un comentario*

*\* de bloque*

*\*/*

* + **COMENTARIOS DE LÍNEA:**

Son comentarios cortos localizados en una sola línea y tabulados al mismo nivel que el código que describen. Si ocupa más de una línea se utilizará un comentario de bloque. Deben estar precedidos por una línea en blanco.

*/\* Esto es un comentario de línea \*/*

*// Esto es otro comentario de línea*

* + **COMENTARIO A FINAL DE LÍNEA**

Comentario situado al final de una sentencia de código y en la misma línea.

*int contador = 4 + 10; // Inicialización del contador*

*int contador = 4 + 10; /\* Inicialización del contador \*/*

## **NOMENCLATURA DE OBJETOS**

El nombramiento de objetos debe cumplir lo siguiente:

* Debe ser claro y conciso específicamente relacionado a los objetos del negocio y la funcionalidad que implementan.
* Se debe evitar el uso de abreviaturas que dificulten la comprensión del código (a excepción que se usen ampliamente como URL o HTML).

## **TIPOS DE PÁGINAS**

* Ayuda: Debe implementarse una página de ayuda explicando la funcionalidad del software o en su defecto todas las páginas del aplicativo.
* Búsqueda: Debe contener los botones Buscar y Cancelar. Debe implementar paginación (por defecto de 10 en 10) en la tabla de resultados. Debe mostrar el total de registros encontrados después de ejecutar la búsqueda. Los campos de búsqueda deben ubicarse aprovechando el espacio horizontal y luego el vertical. En la parte superior derecha de la tabla de resultados debe implementarse el botón “Exportar a Excel”.
* Inicio de sesión: Debe contener el encabezado y el pie de página establecido para todas las páginas de la aplicación.
* Registro de datos: Debe contener los botones Grabar y Cancelar. Los campos deben ubicarse aprovechando el espacio horizontal y luego el vertical. Debe agruparse campos relacionados. Los campos de sólo lectura deben mostrarse como textos o deshabilitados.

## **ESTRUCTURA DE UNA PÁGINA**

* Se debe estructurar la página de acuerdo a: Encabezado, menú, contenido y pie de página.

**Encabezado:**

* Debe contener el nombre de la aplicación (siglas en mayúsculas y el nombre completo en mayúsculas y minúsculas) en la parte superior central.
* Debe contener el logo de la institución en la parte superior izquierda; conservando las dimensiones reguladas por la oficina de comunicaciones.
* Se debe agregar una barra de información que incluya:
  + En la parte derecha: login del usuario que ingresó a la aplicación y la opción para “Cerrar sesión”.

**Menú:**

* Debe ubicarse en la parte superior izquierda, debajo del encabezado.
* Debe mostrar de manera horizontal los sub menú de las opciones del menú principal.

**Contenido:**

* Debe mostrar las páginas que implementan la funcionalidad del aplicativo.
* El contenido de las páginas debe implementar su tamaño en porcentaje.

**Pie de página:**

* Debe ubicarse en la parte inferior de la página.
* Debe contener la lista de navegadores en la cual ha sido probada la aplicación, por ejemplo: Versión optimizada para Internet Explorer, FireFox, etc.
* Debe contener el horario de atención, el teléfono, correo electrónico de soporte informático, la dirección de la organización, el nombre del área de desarrollo de aplicaciones web y el año de implementación, por ejemplo:

Si necesita ayuda o asistencia técnica comuníquese de Lunes a Viernes, de 8:30 am a 5:30 pm a los teléfonos: (01) xxxxx o al correo electrónico:

Calle Aldabas N° 337 – Urb. Las Gardenias Lima – Santiago de Surco - Oficina de Tecnologías de Información (OTI) | Copyright © 2019

## **RECURSOS ESTÁTICOS**

* Los recursos estáticos (imágenes, estilos y javascript) debe ubicarse en la carpeta wwwroot.
* Otros recursos estáticos como documentos relacionados al aplicativo deben ubicarse en el Gestor de Documentos LaserFiche. Si son documentos como Resoluciones deben obtenerse del Gestor de Contenidos del Portal Web utilizando rutas relativas por código de identificación del documento.

**IMÁGENES**

* Se debe crear un sólo folder (carpeta, paquete) con el nombre imagen para contener las imágenes que se utilizarán en todas las páginas del aplicativo.
* Se debe utilizar imágenes con formatos png (cuando se requiera fondo transparente), gif (menor peso y pocos colores) y jpg (mejor calidad y resolución).
* Se debe utilizar imágenes significativas e intuitivas (ejemplo: impresora para imprimir) para expresar la misma funcionalidad en distintas pantallas de la aplicación y evitar imágenes duplicadas que representen lo mismo.
* Se debe crear imágenes considerando los lineamientos de diseño gráfico establecidas por la Oficina de Comunicaciones.

**ESTILOS**

* Se debe crear un sólo folder (carpeta, paquete) con el nombre “css” para contener un solo archivo de estilos que se utilizará en todas las páginas del aplicativo.
* Se debe crear un archivo de estilos por cada dispositivo (navegador web, móvil, otros) donde se visualizará la aplicación.
* El archivo de estilos debe contener atributos relativos al aspecto de la página y no dejar ninguno al código: fuente, controles visuales.

**JAVASCRIPT**

* Se debe utilizar la versión de javascript que sea compatible con todos los navegadores web en versiones recientes (Internet Explorer, Firefox, Chrome).

## **VALIDACIÓN DE ENTRADA**

* Se debe validar el ingreso de datos por el tipo de dato (letras, números enteros o decimales), obligatoriedad, formato específico (correo electrónico, fecha, teléfono, etc.) o rango de valores.
* Se debe mostrar el mensaje de no cumplimiento de validación cuando se presione el botón grabar o buscar.
* Se debe mostrar los campos que no cumplieron la validación con un color rojo y debe enfocarse sobre los campos para ingresar el valor adecuado.
* Se debe controlar el tamaño y la cantidad de datos de los controles de entrada de datos para soportar el tamaño de almacenamiento en los campos de la tabla de base de datos donde persistirán.

## **MENSAJE DE CONFIRMACIÓN, ÉXITO y ERROR**

* Los mensajes de confirmación, éxito y error se deben ubicar en la parte central de todo la pantalla a fin que se fácil de ubicar.
* Se debe implementar mensajes de confirmación para las acciones de grabar, actualizar y eliminar, utilizando oraciones en pregunta, por ejemplo: ¿Está seguro de grabar los datos?.
* Se debe mostrar un mensaje de éxito de color verde, cuando se graba o actualiza datos, por ejemplo: “Se grabó correctamente los datos”. Debe personalizarse el mensaje indicando el nombre del objeto de negocio sobre el cual se ejecutó la acción.
* Se debe mostrar un mensaje de error de color rojo, cuando se interrumpe o existe error durante las acciones de grabar, actualizar, eliminar o buscar datos; por ejemplo: “Hubo un error y no se pudo registrar correctamente la información”.

## **REPORTE**

* Debe contener en la parte superior izquierda el logo de la organización, el título del reporte en el centro y la denominación del año actual en la parte derecha.
* Debe mostrar la fecha de impresión en la parte inferior izquierda y el texto “página actual / total de páginas” en la parte inferior derecha.
* Por defecto, los reportes se deben generar en formato PDF.

## **PAGINACIÓN**

* Se debe implementar un esquema de paginación que evite el tráfico de grandes volúmenes de datos y facilite la navegación.
* Para grandes volúmenes de data, se recomienda realizar la paginación por base de datos utilizando el método basado en el valor (Value based method).
* Por defecto, los resultados de las páginas deben paginarse de 10 en 10.

## **EJECUCIÓN DE CONSULTAS**

* Se recomienda no incrustar sentencias SQL en los métodos de las clases.

## **LLAMADA A PROCEDIMIENTOS ALMACENADOS**

* Se debe ejecutar operaciones de las clases utilizando llamadas a procedimientos almacenados.

## **SEGURIDAD**

## **TRATAMIENTO DE EXCEPCIONES**

* Se debe usar la tecnología base de trazas para capturar y trazar excepciones en un archivo log.
* No se debe mostrar los errores por consola.
* El archivo log debe contener diferentes tipos de información:
  + Todos los eventos de autenticación (inicio de sesión, cierre de sesión, intento de acceso fallido, etc.) que permitan detectar ataques de fuerza bruta y también ataques por adivinación.
  + Información significativa, como el nombre del método y los parámetros recibidos por el método.
* Se debe usar try y varios catch para capturar y trazar todas las excepciones, desde el más específico hasta el menos específico.
* Se debe usar finally para liberar recursos, como conexiones o ficheros.
* Se debe evitar concatenar los mensajes de la excepción, es decir, se debe mostrar toda la excepción.
* Si es necesario personalizar el tratamiento de una excepción, se debe crear una clase que herede de la clase Exception.
* Se debe evitar capturar y trazar excepciones constantemente o la misma excepción en distintos métodos de distintas capas.
* Se debe evitar trazar excepciones en los métodos de la capa de presentación para que no se muestre el error en el navegador.
* Se debe evitar presentar al usuario una traza o información técnica de la excepción ocurrida.
* Se debe evitar la presentación al usuario de los mensajes de error detallados, tales como trazas que devuelven información privada, es decir, se debe utilizar un mensaje de error genérico inclusive para los códigos de respuesta de estado de http (404 o 500).

## **VALIDACIÓN DE DATOS**

La validación apropiada de la entrada de datos a una aplicación evita ataques de inyección, ataques locale/Unicode, ataques al sistema de archivos y desbordamientos de memoria.

* Se debe prevenir la manipulación de parámetros, revisando los mismos antes de utilizarse por primera vez.
* Se debe evitar el uso de campos ocultos para almacenar información sensible.
* Se debe realizar validaciones en el servidor previamente a utilizarlos, siempre que sea necesario debido a que la validación del lado del cliente (Javascript) no es completamente segura y puede ser vulnerada.
* Se debe validar la entrada del usuario a los archivos de registro (histórico, transacciones, archivo log) para evitar su manipulación o la inserción de contenido malicioso.
* Se debe controlar el valor de retorno de los métodos para evitar que un valor devuelto no sea el esperado y no pueda ser controlado.

## **PRUEBAS**

La ejecución de pruebas de software asegura que un aplicativo informático cumpla las especificaciones funcionales y no funcionales requeridas por los usuarios del aplicativo.

* Se debe ejecutar pruebas durante todo el ciclo de vida de la construcción del aplicativo informático y lo más temprano posible para detectar y corregir los defectos y errores aplicando la refactorización del código.

## **TIPOS DE PRUEBAS**

Los tipos de pruebas que se deben aplicar durante todo el ciclo de vida de la construcción del aplicativo informático son:

## **PRUEBA UNITARIA**

La prueba unitaria comprueba que una parte del código funcione de acuerdo a lo solicitado en el requerimiento funcional.

* Se debe implementar pruebas unitarias a la vez que se construye código del aplicativo.
* Si se requiere implementar clases de prueba, se deben ubicar en [modulo] > prueba y los recursos de configuración para ejecutar las pruebas se deben ubicar en [modulo] > prueba > recurso.
* Se debe retirar las clases de prueba de la aplicación cuando se despliega en ambiente de producción.

## **PRUEBA DE INTEGRACIÓN**

La prueba de integración se ejecuta después que se aprobaron las pruebas unitarias con el objetivo de comprobar que todas las partes del código funcionen en conjunto y de acuerdo a lo solicitado en el requerimiento funcional.

* Si se requiere implementar clases de prueba, se deben ubicar en [modulo] > prueba y los recursos de configuración para ejecutar las pruebas se deben ubicar en [modulo] > prueba > recurso.
* Se debe retirar las clases de prueba de la aplicación cuando se despliega en ambiente de producción.

## **PRUEBA DE ACEPTACIÓN**

La prueba de aceptación permite al usuario final comprobar que todos los requerimientos funcionales de la aplicación estén implementados.

* Si se requiere implementar clases de prueba, se deben ubicar en [modulo] > prueba y los recursos de configuración para ejecutar las pruebas se deben ubicar en [modulo] > prueba > recurso.
* Se debe retirar las clases de prueba de la aplicación cuando se despliega en ambiente de producción.

## **PRUEBA DE SERVICIOS WEB**

La prueba de servicios web permite comprobar su correcto funcionamiento y automatizar dichas pruebas para repetirlas en cualquier momento.

* Si se requiere implementar clases de prueba, se deben ubicar en [modulo] > prueba y los recursos de configuración para ejecutar las pruebas se deben ubicar en [modulo] > prueba > recurso.
* Se debe retirar las clases de prueba de la aplicación cuando se despliega en ambiente de producción.

## **PRUEBA DE CARGA Y STRESS**

La prueba de carga y stress evalúa el comportamiento y rendimiento del aplicativo en situaciones extremas, tanto en carga como en tiempo, donde los usuarios y peticiones se producen de forma concurrente.

* Si se requiere implementar clases de prueba de carga y stress se deben ubicar en [modulo] > prueba > jmeter y los recursos de configuración para ejecutar las pruebas se deben ubicar en [modulo] > prueba > recurso.
* Se debe retirar las clases de prueba de la aplicación cuando se despliega en ambiente de producción.

## **ANÁLISIS DE CÓDIGO ESTÁTICO**

El análisis de código estático es la revisión del código por el propio programador que se ejecuta frecuentemente.

## **PLAN DE PRUEBAS**

El Plan de Pruebas es un documento que describe el alcance, enfoque, recursos y calendario de actividades de prueba.

* Se debe elaborar un plan de pruebas por el aplicativo o por cada módulo del aplicativo.
* El plan de pruebas debe incluir los tipos de pruebas, los datos utilizados y el resultado.

## **ENTREGA**

* Se debe versionar el código fuente del aplicativo informático desde el inicio de su construcción utilizando la herramienta de Control de Versiones existente.
* Se debe realizar entregas parciales potencialmente funcionales para ser presentadas al usuario.
* Al terminar la construcción y pruebas del aplicativo informático se debe entregar las fuentes, los compilados y scripts necesarios utilizando el formato de Solicitud de Pase a Producción.

## **TIPOS DE ARCHIVOS QUE SE UTILIZAN EN UNA APLICACIÓN WEB**

En la siguiente tabla se describen los tipos de archivos implementados en una aplicación Web.

| Tipo de archivo | Extensiones de archivo | Descripción |
| --- | --- | --- |
| Estructurales | HTML | En esta categoría se incluyen los archivos que describen lógicamente las interfaces de los objetos que componen una aplicación web. |
| Funcionales (cliente) | js, css, vs | Son archivos que implementan funcionalidades o apariencia de la aplicación web desde el lado del cliente. |
| Imágenes | gif, jpg, bmp | Son archivos binarios de imágenes en varios formatos. |
| Documentos | pdf, ps, doc, xls, ppt, rtf | Este tipo de archivos, comprende los documentos que pueden ser desplegados o descargados de la aplicación web por el cliente mediante el navegador. |

1. **Estados:**

   E: Indica Elaborado por

   M: Indica Modificado por

   R: Indica Revisado por

   A: Indica Aprobado por [↑](#footnote-ref-1)