Manual de estándares de programación y buenas prácticas

Alumnos:

* José Sandoval
* Roy Taza
* Aldo Rodríguez
* Rodrigo Canales

**ÍNDICE**

[**I.** **INTRODUCCIÓN** 3](#_Toc444468193)

[**1.** **Objetivos** 3](#_Toc444468194)

[**2.** **Alcance** 3](#_Toc444468195)

[**3.** **Definiciones, Acrónimos, y Abreviaturas** 3](#_Toc444468196)

[**II.** **CONVENCIONES Y ESTÁNDARES DE CODIFICACION** 4](#_Toc444468197)

[**1.** **Convenios de Capitalización** 4](#_Toc444468198)

[**2.** **Sangría y espaciamiento** 9](#_Toc444468199)

[**3.** **Buenas prácticas de Programación** 12](#_Toc444468200)

[**4.** **Comentarios** 19](#_Toc444468201)

[**5.** **Manejos de Errores y Excepciones** 21](#_Toc444468202)

[**III.** **CODIFICACION HTML, JavaScript, CSS** 23](#_Toc444468203)

[**1.** **HTML CSS** 23](#_Toc444468204)

[**2.** **JavaScript** 27](#_Toc444468205)

[**IV.** **CODIFICACION MVC.NET** 30](#_Toc444468206)

[**1.** **Reglas Generales** 30](#_Toc444468207)

[**BIBLIOGRAFIA** 37](#_Toc444468208)

1. **INTRODUCCIÓN**
2. **Objetivos**

Describir y definir un estándar de programación y código para mantener un buen nivel de cumplimiento de buenas prácticas, durante la construcción del proyecto. Asimismo, servirá para realizar reuniones de revisión de código para asegurar que cada uno está siguiendo las reglas.

1. **Alcance**

En este documento se describan los estándares MVC, C#, HTML, JavaScript y demás Frameworks a usar; que permitirá tener un estándar de escritura de código sobre el proyecto y ayudara a el mantenimiento del sistema de forma integral.

1. **CONVENCIONES Y ESTÁNDARES DE CODIFICACION**

Microsoft, tiene documentación acerca de las buenas prácticas de desarrollo en su canal MSDN, las cuales serán tomadas en cuanta durante el documento. Por ello es necesario describir las diferentes notaciones que se maneja a nivel de la compañía propietaria y creadora de la herramienta y Framework de desarrollo que el grupo utilizará durante la construcción del sistema.

Las notaciones utilizadas por Microsoft son las siguientes.

1. **Convenios de Capitalización**

Los convenios de capitalización que se describen hacen que sea fácil para los desarrolladores entender y trabajar.

Los siguientes términos describen diferentes formas de identificadores.

* **Pascal Casing (Notación Pascal)**

La primera letra en el identificador y la primera letra de las siguientes palabras concatenadas aparecen en mayúsculas. Puede utilizar el caso de Pascal para los identificadores de tres o más caracteres. Por ejemplo:

BackColor

* **Camell Casing(Notación Camell)**

La primera letra de un identificador es minúscula y la primera letra de cada palabra se escribe con mayúscula concatenadas. Por ejemplo:

backColor

* **Uppercase**

Todas las letras en el identificador se capitalizan. Por ejemplo:

BACKCOLOR

* **Lowercase**

Todas las letras en el identificador se mantienen en minúscula y separados por guion (-). Por ejemplo:

back-color

1. Usar notación Pascal para el nombre de las Clases.

public class LoginViewModel

{

...

}

1. Usar notación Pascal para el nombre de los Métodos.

void DiHola(String Nombre)

{

...

}

1. Usar notación de Camell para variables y Pascal para parámetros de los métodos.

Int32 cuentaTotal = 0;

void DiHola(String Nombre)

{

String mensajeCompleto = “Hola” + Nombre;

(...)

}

1. Usar el sufijo “I” con notación Pascal para las interfaces (Ejemplo: IEntity). Utiliza “T” para estructuras de tipos de datos.

public class TFactura {

public String NombreDelCliente;

(...)

public ArrayList LineasDeDetalle = new ArrayList();

}

1. No usar notación Húngara para el nombre de las variables.

En épocas previas muchos de los programadores les agradaba la notación Húngara – la cual especifica el tipo de dato de la variable como un prefijo en el nombre y usar el prefijo m\_ para variables globales. Por Ejemplo:

String m\_strNombre;

Int32 intEdad;

Sin embargo, en los estándares de codificación de .NET, esto no es recomendado. El uso del tipo de dato y del prefijo m\_ para representar variables globales no debe ser usado. Todas las variables deben usar una notación Camell.

Aun así algunos programadores prefieren usar el prefijo m\_ para representar variables globales dado que no hay otra forma fácil de identificar una variable global.

String nombre;

Int32 edad;

1. Usar palabras entendibles el prefijo (\_) para variables privadas. No usar guiones bajos (\_) para nombres de variables locales

public class Login

{

private String \_usuario;

private String \_password;

...

}

1. Usar palabras entendibles y descriptivas para nombrar a las variables. NO USAR ABREVIACIONES.

Correcto:

Int32 usuarioId;

Int32 cantidadUsuarios;

Decimal salario;

String direccion;

Incorrecto:

Int32 id;

Int32 cantUsuario;

Decimal sal;

String dir;

1. No usar nombres de variables de un solo carácter como i, a, n, s etc.

Usar nombres como:

indice, temp, aux

Una excepción en este caso podría ser las variables usadas para iteraciones:

for ( int i = 0; i < cantidad; i++)

{

...

}

Si la variable es usada solo como un contador para una iteración y no es usada en ningún otro lado dentro del ciclo.

1. Todas variables globales deben usar la notación Uppercase y con guion bajo (\_) como separación de palabras de tal forma que puedan ser identificadas de otras variables locales.

public static class ConstantHelpers

{

public const Int32 PAGE\_SIZE = 10;

...

}

1. No usar palabras reservadas para nombres de variables.
2. Usar el prefijo “Es” el “Is” para variables de tipo Booleano prefijos similares.

Ejemplo:

public static Boolean IsLoggedIn()

{

return Get(Session, SessionKey.Rol) != null;

}

public Boolean EsValido;

public Boolean IsActive;

La propuesta de “Is”, más aceptada, tiene coherencia con las propiedades de uso global de .NET, por ejemplo: objeto.IsEmpty(); ModelState.IsValid(); se agrupa en el IntelliSense de una forma más coherente.

1. Los nombres de los espacios de nombres deben seguir el siguiente estándar de patrón

<NombreDeCompañía>.<NombreDeProducto>.<MóduloSuperior>.<MóduloInferior>

namespace UPC.CA.Presentation.Controllers.Home

1. Los nombres de Clases o Métodos y funciones deben seguir el estándar:

<Accion/Verbo en Inglés><descripción>

ViewCliente(Int32? ClienteId);

ListClientes();

AddEditCliente(Int32? ClienteId);

RemoveCliente(Int32? ClienteId);

De esta manera se agrupará mejor en el IntelliSense con el resto de métodos generales de los objetos.

1. El nombre de los archivos debe coincidir con el nombre de la clase.

Por ejemplo, para la clase LoginViewModel el nombre del archivo debe ser LoginViewModel.cs . Y usa notación Pascal para el nombre de los archivos.

1. Para agregar métodos de extensión deberán estar en idioma Ingles.

**public** static void SaveAsXML**(this** **object** Obj**,** String FileName**)**

**{**

var xmlRequest **=** **new** StreamWriter**(F**ileName**);**

var xmlFileRequest **=** **new** XmlSerializer**(**obj**.**GetType**());**

xmlFileRequest**.**Serialize**(**xmlRequest**,** Obj**);**

xmlRequest**.**Close**();**

**}**

1. **Sangría y espaciamiento**
2. Usa TAB para la sangría. No usar ESPACIOS. Recomendación, Usar el *code formating* de la Herramienta Visual Studio con las teclas rápidas: Ctrl + K + D.
3. Los comentarios deben estar al mismo nivel que el código (usar el mismo nivel de sangría).

**Correcto:**

//Formatea un mensaje y lo despliega

String mensajeCompleto **=** "Hola" **+** nombre**;**

DateTime horaActual **=** DateTime**.**Now**;**

String mensaje **=** mensajeCompleto **+** "**,** la hora es**:** " **+** horaActual**.**ToShortTimeString**();**

return**(**mensaje**);**

**Incorrecto:**

//Formatea un mensaje y lo despliega

String mensajeCompleto **=** "Hola" **+** nombre**;**

DateTime horaActual **=** DateTime**.**Now**;**

String mensaje **=** mensajeCompleto **+** "**,** la hora es**:** " **+** horaActual**.**ToShortTimeString**();**

return**(**mensaje**);**

1. Las llaves ({}) deben estar en el mismo nivel que el código fuera de las llaves.

**Correcto:**

**if** **(** condicion **)**

**{**

//Haz algo

**return** **false;**

**}**

1. Usar una línea en blanco para separar un grupo lógico de código

**Correcto:**

Boolean DiHola **(**String Nombre**)**

**{**

String mensajeCompleto **=** "Hola" **+** Nombre**;**

DateTime horaActual **=** DateTime**.**Now**;**

String mensaje **=** mensajeCompleto **+** "**,** la hora es**:** " **+** horaActual**.**ToShortTimeString**();**

MessageBox**.**Show**(**mensaje**);**

**if** **(** … **)**

**{**

//Haz algo

// …

**return** **false;**

**}**

**return** **true;**

**}**

**Incorrecto:**

Boolean DiHola **(**String Nombre**)**

**{**

String mensajeCompleto **=** "Hola" **+** Nombre**;**

DateTime horaActual **=** DateTime**.**Now**;**

String mensaje **=** mensajeCompleto **+** "**,** la hora es**:** " **+** horaActual**.**ToShortTimeString**();**

MessageBox**.**Show**(**mensaje**);**

**if** **(** … **)** **{**

//Haz algo

// …

**return** **false;**

**}**

**return** **true;**

**}**

1. Debe haber una y solo una línea en blanco entre cada método dentro de las Clases.

1. Las llaves ({}) deben estar en una línea separada y no en la misma línea del if, for, switch etc.

**Correcto:**

**if** **(** condicion **)**

**{**

//Haz algo

// …

**return** **false;**

**}**

**Incorrecto:**

**if** **(** condicion **) {**

//Haz algo

// …

**return** **false;**

**}**

1. Usa un espacio simple antes y después de los paréntesis y los operadores.

**Correcto:**

**if** **(** muestraResultado **==** **true** **)**

**{**

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

**{**

//

**}**

**}**

**Incorrecto:**

**if(**muestraResultado **==** **true)**

**{**

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

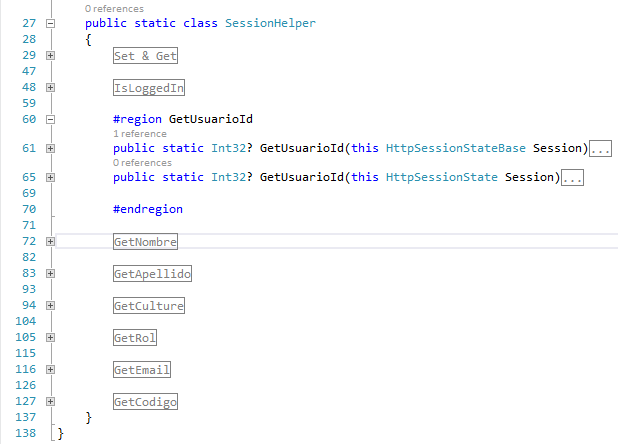
**{**

//

**}**

**}**

1. Usa #region para agrupar piezas de código juntas. Si se usa una agrupación apropiada usando #region, la página debe verse como a continuación cuando todas las definiciones estén cerradas.



1. **Buenas prácticas de Programación**
2. El nombre de los métodos debe decir lo que hace. No usar nombres engañosos. Si el nombre del método es obvio, no hay necesidad de que la documentación que explique qué hace el método.

**Correcto:**

void GuardarDatosUsuario **(** Int32 UsuarioId**)**

**{**

**...**

**}**

**Incorrecto:**

//Este método guardará los datos de usuario

void GuardarDatos **(** Int32 UsuarioId**)**

**{**

**...**

**}**

1. Un método debe tener solo una tarea. No combinar más de una tarea en un solo método, aún si esas tareas son pequeñas.

**Correcto:**

//Guarda la dirección

//Envía un email para informar que la dirección es actualizada.

SetDireccion**(** direccion **);**

EnviaCorreoAlSupervisor**(** direccion**,** email **);**

void SetDireccion **(** String Direccion **)**

**{**

// Guarda la dirección.

// ...

**}**

void EnviaCorreoAlSupervisor **(** String Direccion**,** String Email **)**

**{**

// Envía un correo para informar que la dirección fue cambiada.

// ...

**}**

**Incorrecto:**

// Gurada la dirección y envía un correo al supervisor para informar que la dirección fue cambiada.

GuardaDireccion **(** direccion**,** email **);**

void GuardaDireccion **(** String Direccion**,** String Email **)**

**{**

// Tarea 1.

// Guarda la dirección.

// ...

// Tarea 2.

// Envía un correo para informar al supervisor que la dirección fue cambiada

// ...

**}**

1. Siempre verifica valores inesperados. por ejemplo, si estas usando un parámetro con dos posibles valores, nunca asumas que si uno no concuerda entonces la única posibilidad es el otro valor.

**Correcto:**

**if** **(** tipoMiembro **==** ConstantHelpers**.**MIEMBRO**.**REGISTRADO **)**

**{**

// Usuario registrado… haz algo…

**}**

**else** **if** **(** tipoMiembro **==** ConstantHelpers**.**MIEMBRO**.**INVITADO **)**

**{**

// Usuario invitado… haz algo…

**}**

**else**

**{**

// Usuario inesperado. Lanza una excepción

**throw** **new** Exception **(**"Valor inesperado" **+** tipoMiembro**.**ToString**()** **+** "**.**" **)**

// Si agregamos un nuevo tipo de usuario en el futuro, podremos fácilmente encontrar el problema aquí.

**}**

**Incorrecto:**

**if** **(** tipoMiembro **==** ConstantHelpers**.**MIEMBRO**.**REGISTRADO **)** **{**

// Usuario registrado… haz algo…

**}**

**else** **if** **(** tipoMiembro **==** ConstantHelpers**.**MIEMBRO**.**INVITADO **)** **{**

// Usuario invitado… haz algo…

// Si nosotros introducimos otro tipo de usuario en el futuro este código fallará y no se notará

**}**

1. No incrustes cadenas de texto en el código. Usa archivos de recursos tales como Web.config .

**public** String GetAbsoluteURL**()**

**{**

**return** ConfigurationManager**.**AppSettings**[**"AbsoluteURL"**];**

**}**

1. Convierte las cadenas de texto a minúsculas o mayúsculas antes de compararlas. Esto asegurará que la cadena coincida.

**if** **(** nombre**.**ToUpper**()** **==** "JUAN" **)**

**{**

// Haz algo

**}**

1. Usa String.IsNullOrEmpty**()** en vez de "" o **null**.

**Correcto:**

**if** **(** String**.**IsNullOrEmpty**(**nombre**)** **)**

**{**

// Haz algo

**}**

**Incorrecto:**

**if** **(** nombre **==** "" **)**

**{**

// Haz algo

**}**

**if** **(** nombre **==** **null** **)**

**{**

// Haz algo

**}**

1. Evita usar variables globales. Declara variables locales siempre que sea necesario y pásalas a otros métodos en vez de compartir una variable global entre métodos. Si compartes una variable global entre métodos, te será difícil rastrear qué método cambia el valor y cuando.
2. Usa enum dondequiera que sea requerido. No usar números o cadenas para indicar valores discretos.

**Correcto:**

enum AppRol **{**

Administrador**,**

Coordinador**,**

Usuario

**}**

void SwitchDashboard **(**AppRol Rol**)**

**{**

**switch** **(** Rol **)**

**{**

**case** AppRol**.**Administrador**:**

// Haz algo

**break;**

**case** AppRol**.**Coordinador**:**

// Haz algo

**break;**

**case** AppRol**.**Usuario**:**

// Haz algo

**break;**

**default:**

// Haz algo

**break;**

**}**

**}**

**Incorrecto:**

void SwitchDashboard **(**String Rol**)**

**{**

**switch** **(**Rol**)**

**{**

**case** "Administrador"**:**

// Haz algo

**break;**

**case** "Coordinador"**:**

// Haz algo

**break;**

**case** "Usuario"**:**

// Haz algo

**break;**

**default:**

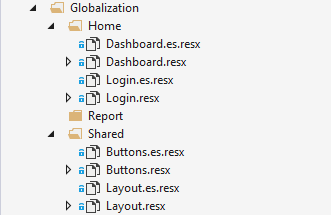
// Haz algo

**break;**

**}**

**}**

1. No incrustar en el código rutas o letras de dispositivos. Obtén la ruta de la aplicación programáticamente y usa rutas relativas a ella.
2. No asumir que el código se ejecutará desde el disco “C:\”. No se sabe si algunos usuarios lo ejecutaran desde la red o desde la unidad “Z:\”.
3. En el arranque de la aplicación, ejecuta una clase de “auto verificación” y asegúrate que todos los archivos requeridos y las dependencias estén disponibles en las ubicaciones esperadas. Verifica las conexiones a la base de datos en el arranque, si es requerido. Dale un mensaje amigable al usuario en caso de algún problema.
4. Los mensajes de error deben ayudar al usuario a resolver el problema. Nunca muestres mensajes de error como “Error en la Aplicación”, “Hay un error…” etc. Cuando se desplieguen los mensajes de error, adicionalmente decirle al usuario qué está mal, el mensaje decirle también lo que el usuario debe hacer para resolver el problema. Por ello dé mensajes específicos como “Fallo al actualizar la base de datos”, sugiera al usuario que debe hacer: “Fallo al actualizar la base de datos. Por favor asegúrate de que la cuenta y la contraseña sean correctos”.
5. Muestra mensajes cortos y amigables al usuario. Pero registra el error actual con toda la información posible. Esto ayudará mucho a diagnosticar problemas.
6. Usar ficheros de recursos (**.resx**) para los literales de la aplicación. Favorece el cambio idiomático, cambio de literales rápido, unificación de mensajes de eventos, etc.



Como se puede apreciar en la imagen anterior, el aplicativo tiene la capacidad de soportar los idiomas español e inglés.

1. No guardar más de una clase en un solo archivo.
2. Evita tener archivos muy grandes. Si un solo archivo tiene más de 1000 líneas de código, es un buen candidato para refactorizar. Dividir lógicamente en dos o más clases.
3. Evita pasar muchos parámetros a un método. Si tienes más de 4~5 parámetros, es un buen candidato para definir una clase o una estructura. Lo contrario destroza el consumo en memoria, hay más facilidad de corrupción de datos, el procesador se ve afectado directamente en el rendimiento... etc.
4. Si tienes un método que retorna una colección, devuelve una colección vacía en vez de null, si no hay datos que retornar. Por ejemplo, si tienes un método que retorna un List<**object**>, siempre retorna un List<**object**> válido. Si no tienes elementos que devolver, entonces retorna un List<**object**> válido con 0 elementos. Esto hará fácil para la aplicación que llama al método verificar solamente la propiedad “Count()” en vez que hacer verificaciones adicionales para “null”.

**public** class SetSedeViewModel **{**

**public** List**<**Sede**>** ListSede **{** get**;** set**;** **}**

**public** SetAgenciaViewModel**()**

**{**

ListSede **=** **new** List**<**Sede**>();**

**}**

**public** void CargarDatos**(**CargarDatosContext dataContext**)**

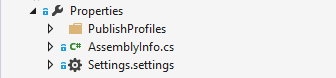
**{**

//Haz Algo

**}**

**}**

1. Usar el archivo AssemblyInfo.cs para llenar la información como el número de versión, descripción, nombre de la compañía, nota de derechos reservados etc.



1. Organizar lógicamente los archivos dentro de carpetas apropiadas. Usar una jerarquía de carpetas de 2 niveles. Puedes tener hasta 10 carpetas en raíz y cada carpeta hasta 5 subcarpetas. Si tienes muchas carpetas que no pueden ser acomodadas en la jerarquía de 2 niveles mencionada arriba, necesitas refactorizar en distintos ensamblados.
2. Declarar variables tan cerca como sea posible de donde son usadas por primera vez. Usar una declaración de variable por línea.

**Incorrecto**

Int32 usuarioId, cantidadUsuarios, salario, direccion;

**Correcto**

Int32 usuarioId;

Int32 cantidadUsuarios;

Decimal salario;

String direccion;

1. **Comentarios**
2. Escribir comentarios donde sea requerido. Tener en cuenta que un buen código legible requerirá de pocos comentarios. Si todas las variables, métodos, constantes tienen nombres significativos, esto hará que el código sea fácilmente legible y no requiera de la introducción de comentarios en exceso.
3. No escribir comentarios en porciones en las que el código es fácilmente entendible y no requiere de conocimiento específico para su comprensión.
4. Utilizar // o /// para introducir comentarios. Evitar usar /\* ... \*/ con comentarios de bloque.
5. Usar bloques de comentarios (///) para funciones con declaración public, protected, e internal.
6. Agregar la documentación del encabezado de cada método (utilizando ///), especificando correctamente:

* El propósito del método
* Nombre, Tipo de datos y Contenido de cada parámetro recibido
* El propósito de la información retornada
* Esto es particularmente útil para generar documentación de forma automática, a partir de estos comentarios.

/// <summary>

/// Convierte una fecha UTC a un string de una fecha "Local" con formato. Se toma la hora de Perú (GMT-5)

/// </summary>

/// <param name="val">Fecha, en UTC</param>

/// <param name="format">Formato para del string resultante</param>

/// <returns>Fecha, en hora de Perú</returns>

**public** static String ToLocalDateString**(this** DateTime Val**,** String Format **=** "dd/MM/yyyy"**)**

**{**

**return** Val**.**ToLocalDate**().**ToString**(**Format**);**

**}**

1. Si se inicializa una variable a un número especial distinto de 0, -1, String.Empty o Null, se deberá documentar la razón por la que se ha escogido dicho valor.
2. Dentro de los comentarios incluir una lista de tareas a realizar durante la asignación bajo la siguiente estructura.

* Tarea a realizar
* Tareas pendientes
* Lo que hace es código hasta el cierre del avance
* Fecha de asignación
* Nombre del responsable o responsables

/// <summary>

/// Convierte una fecha UTC a un String de una fecha "Local" con formato. Se toma la hora de Perú (GMT-5)

/// TODO: Convierte una fecha UTC a un String de una fecha "Local"

/// UNDONE:

/// HACK: Temporalmente guarda desde un String mas no usando la configuración de cultura del aplicativo

/// BEGIN DATE: 28/01/2016

/// ASIGNED TO: Jonathan De La Cruz

/// </summary>

/// <param name="val">Fecha, en UTC</param>

/// <param name="format">Formato para del string resultante</param>

/// <returns>Fecha, en hora de Perú</returns>

**public** static String ToLocalDateString**(this** DateTime Val**,** String Format **=** "dd/MM/yyyy"**)**

**{**

**return** Val**.**ToLocalDate**().**ToString**(**Format**);**

**}**

Esto ayudara a crear la documentación necesaria del código de manera automática por el Visual Studio.

1. **Manejos de Errores y Excepciones**
2. Nunca “capturar una excepción y no hacer nada”. Si se oculta una excepción, nunca se sabrá si la excepción sucedido. Muchos desarrolladores utilizan este método para ignorar errores no significativos. Tratar de evitar las excepciones comprobando todas las condiciones de error mediante programación. En el peor de los casos, se deberá registrar la excepción en un log y continuar.
3. En casos de excepción, mostrar un mensaje amigable al usuario, pero se debe registrar la excepción actual con todos los detalles posibles sobre el error dentro de un log, incluyendo el momento en que ocurrió, el método y nombre de la clase, etc. Siempre capturar una excepción específica, no genéricas. Cuando vuelva a lanzar una excepción, utilice la instrucción throw sin especificar la excepción original.

**try**

**{**

//Haz algo

**}**

**catch** **(**Exception**)**

**{**

//Registrar Error

**throw;**

**}**

1. Cada **namespace**, clase subclase deberá tener un número identificador asignado el cual será guardado dentro de un log de errores con datos resaltantes para su resolución.
2. No escribir **try** **catch** en todos los métodos. Utilícelo sólo si hay una posibilidad de que una excepción específica se puede producir y no se puede evitar por cualquier otro medio.
3. Al abrir conexiones a una base de datos, sockets, archivos etc., cerrar las conexiones con el bloque **finally**. Esto asegurará que aún si una excepción ocurre después de abrir la conexión, se cerrará seguramente en el bloque **finally**.

Int32 numero **=** 123**;**

String texto **=** "mensaje de texto"**;**

**object** objeto **=** s**;**

**try**

**{**

**using (**var transaction = new TransactionScope();**)**

**{**

// Invalid conversion; o contains a string not an int numero = (int) objeto;

**}**

**}**

**finally**

**{**

Console**.**Write**(**"Número = {0}"**,** numero**);**

**}**

1. **CODIFICACION HTML, JavaScript, CSS**

Para el siguiente modulo se tomara en cuenta el framework Bootstrap, por ello los siguiente puntos se enfocaran en dar reglas para seguir con este estándar, así como un buen uso de las etiquetas HTML5

1. **HTML CSS**
2. Colocar las referencias a CSS en el <head>, y las referencias JavaScript al final: dado que el contenido y el diseño vienen definidos en distintos documentos, puede darse el caso de que el navegador muestre a nuestros visitantes lo primero antes que lo segundo. Para intentar reducir al mínimo esa posibilidad, lo aconsejable es poner las llamadas a CSS lo antes posible, en el encabezado del documento HTML, con el fin de que el navegador acceda a ellas desde el primer momento. Del mismo modo, ganaremos en sensación de velocidad para el cliente si dejamos las referencias a archivos JavaScript al final, para que la web se pueda representar incluso antes de que éstos se ejecuten.

<!DOCTYPE html>

<html lang=**"en"**>

<head>

<meta charset=**"utf-8"**>

<meta http-equiv=**"X-UA-Compatible"** content=**"IE=edge"**>

<meta name=**"viewport"** content=**"width=device-width, initial-scale=1"**>

<title>**Sistema de Acre…..ditación**</title>

<!-- Bootstrap -->

<link href=**"css/bootstrap.min.css"** rel=**"stylesheet"**>

<!-- Fin Bootstrap -->

<!-- CSS -->

<link href=**"css/style.css"** rel=**"stylesheet"**>

<link href=**"css/custom-styles.css"** rel=**"stylesheet"**>

<!-- Fin CSS -->

</head>

<body>

<!-- Contenido -->

<h1>**Hola, mundo!**</h1>

**...**

<!-- Fin Contenido -->

<!-- Scripts -->

<script src=**"https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"**></script>

<script src=**"js/bootstrap.min.js"**></script>

<script src=**"js/index.js"**></script>

<!-- Fin Scripts -->

</body>

</html>

1. Siempre cerrar las etiquetas, un documento HTML está compuesto por muchas etiquetas que sirven para definir encabezados, párrafos e incluso secciones enteras, usualmente vienen en pares, una para abrir y otra para cerrar. Siempre que abras una etiqueta, recuerda cerrarla correctamente.

**Correcto**

<p>**Lorem ipsum dolor sit amet**</p>

**Incorrecto**

<p>**Lorem ipsum dolor sit amet**

1. Nombrar las etiquetas siempre en minúsculas (lowercase), es probable que los navegadores modernos no interpreten correctamente estas etiquetas.

**Correcto**

<p>**Lorem ipsum dolor sit amet**</p>

<div>

**Lorem ipsum dolor sit amet**

</div>

**Incorrecto**

<P>**Lorem ipsum dolor sit amet**</P>

<DIV>

**Lorem ipsum dolor sit amet**

</DIV>

1. Utilizar ficheros externos para CSS y JavaScript, Aunque se puede crear funciones JavaScript y dar estilo a los elementos directamente en el documento HTML, lo que es esperado evitar hacerlo con ciertas excepciones. Utiliza hojas de estilo y *scripts* externos en archivos separados para estos fines.

1. Usa el atributo **“alt”** en todas las imágenes Es una regla fácil de olvidar, sin embargo hay que hacer un esfuerzo por recordar incluir el atributo **“alt”** en todas las imágenes, es necesario incluir este atributo por razones de validación y accesibilidad.

<img src=**"/**Content/img/logo.png" **ondragstart="**return false;**" alt="**logo" />

1. Usar comillas dobles para definir parámetros de las etiquetas HTML con acciones JavaScript.

<a href=**"#**" **onclick="**alert('Iniciar')**"**/> Iniciar </a>

1. Nombrar los id’s, nombres y clases en notación lowercase.

**Correcto**

<input type=**"text**" class=**"form-control**" id=**"nombre-usuario**" name=**"nombre-usuario**" placeholder=**"Nombre**"/>

<form method=**"post**" class=**"form**" id=**"form-usuario**" >

**...**

</form>

**Incorrecto**

<input type=**"text**" class=**"form-control**" id=**"Nombre-Usuario**" name=**"Nombre-Usuario**" placeholder=**"Nombre**"/>

<form method=**"post**" class=**"form**" id=**"Form-Usuario**" >

**...**

</form>

1. Usar comentarios para separar funcionalidades y agrupar etiquetas que tengas una determinada funcionalidad. De manera que se pueda conocer que hace la sección de la vista HTML.

<!-- Formulario Registro de Usuario -->

<form method=**"post**" class=**"form**" id=**"form-usuario**" >

**...**

</form>  
<!-- Fin Formulario Registro de Usuario -->

1. Separar bien el contenido y diseño: el código CSS puede implementarse desde un archivo externo, incrustado en la cabecera del documento o directamente en las propias etiquetas HTML. Esta última opción rompe por completo el concepto de separación entre contenido y diseño, convirtiéndose en una práctica a evitar por la gran cantidad de problemas que puede suponer.

<!DOCTYPE html>

<html>

<head>

<title>**Sistema de Acreditación - Login**</title>

<!-- Bootstrap -->

<link href=**"css/bootstrap.min.css"** rel=**"stylesheet"**>

<!-- Fin Bootstrap -->

<!-- CSS -->

<link href=**"css/style.css"** rel=**"stylesheet"**>

<link href=**"css/login.css"** rel=**"stylesheet"**>

<link href=**"css/custom-styles.css"** rel=**"stylesheet"**>

<!-- Fin CSS -->

</head>

<body>

**...**

</body>

</html>

1. Usar los prefijos de proveedor: en ocasiones los navegadores ofrecen compatibilidad con estándares que aún están pendientes de aprobación. Para poder hacer uso de los mismos, disponemos de prefijos que añadidos a cada propiedad nos permiten probarlos de forma confiable, atajando además posibles problemas futuros de compatibilidad.

**.**bg-gradiente **{**

**text-align:center; width: 500px; height: 9em;**

**line-height: 4.5em; margin: 0 auto;**

**border: solid 3px;**

**background-color: #F4FA58;** /\*Navegadores que no aceptan el gradiente de fondo\*/

**background-image: -webkit-gradient(linear, left top, left bottom, from(#F4FA58), to(#FF0000));** /\* Chrome, Safari versiones más antiguas \*/

**background-image: -webkit-linear-gradient(top, #F4FA58, #FF0000);** /\* Chrome, Safari versiones relativamente modernas \*/

**background-image: -moz-linear-gradient(top, #F4FA58, #FF0000);** /\* Firefox versiones relativamente modernas\*/

**background-image: -o-linear-gradient(top, #F4FA58, #FF0000);** /\* Opera versiones relativamente modernas\*/

**background-image: linear-gradient(to bottom, #F4FA58, #FF0000);** /\* Chrome, Firefox, IE, Opera versiones actuales \*/

**}**

1. Evitar repeticiones, otro concepto bastante básico pero que siempre conviene recordar, pues acabar con las redundancias en nuestra hoja de estilos agrupando cuando sea posible, hará las cosas más fáciles y ayudará además a aligerar la carga del documento en buena medida.
2. Asignar Nombres en lowercase separados por guion (**-**) para archivos CSS:

**custom-styles.css**

1. Asignar Nombres en lowercase separados por puntos (**.**) en para archivos JavaScript:

**view.reporte.js**

1. **JavaScript**

En la siguiente sección se tomara en cuenta las buenas prácticas de JavaScript y JQuery, con esta tecnología también aplica las reglas de identación definidas en el segundo punto del documento.

1. Usar **$(document).ready()**de forma tal que el bloque se ejecutará sólo una vez que la página esté disponible.

$**(document).**ready**(function()** **{**

// Ingresar lógica en esta sección

**});**

1. Utilizar **===** en vez de **==**. JavaScript utiliza dos tipos diferentes de operadores de igualdad: **===** | **!==** y **==** | **!=**.
2. Usar comillas simples ('') para asignar cadenas de texto en vez de comillas dobles ("") para el caso de JavaScript.

**<**script**>**

**var** fechaHoy **=** **new** Date**();**

**document.**write**(**'Hoy es día <strong class="text-info">'**,** fechaHoy**.**getDate**(),** '</strong>'**);**

**</**script**>**

1. Declara las variables fuera de la sentencia **for** Cuando ejecute sentencias **for** muy extensos, no haga que el motor trabaje mucho.

**Incorrecto**

**for(var** i **=** 0 **;** i <**=** algunArray**.**length**;** i**++)** **{**

**var** contenedor **=** **document.**getElementById**(**'contenedor'**);**

contenedor**.**innerHtml **+=** 'mi numero: ' **+** i**;**

console**.**log**(**i**);**

**}**

**Correcto**

**var** contenedor **=** **document.**getElementById**(**'contenedor'**);**

**var** longitud **=** algunArray**.**length**;**

**var** contenido **=** ''**;**

**for( var** i **=** 0**;** i **<=** longitud**;** i**++)** **{**

contenido **+=** 'mi numero: ' **+** i**;**

console**.**log**(**i**);**

**}**

contenedor**.**innerHtml **=** contenido**;**

1. Agrupar las operaciones JQuery, no es necesario que los selecciones uno por uno. Si los agrupas utilizando un único selector, irás más rápido y el código será más corto.

**Incorrecto**

$**(**'#formularioCrear'**).**show**();**

$**(**'#tablaItems'**).**show**();**

$**(**'#mensajeAlerta'**).**show**();**

**Correcto**

$**(**'#formularioCrear, #tablaItems, #mensajeAlerta'**).**show**();**

1. Encadenar la Cache de selecciones. Lo ideal es no abusar demasiado de las búsquedas en el DOM y guardar el resultado en un variable, o bien aprovechar el encadenamiento de funciones de JQuery.

$**(**'a:not([href\*=upc.edu.pe])'**).**attr**(**'target'**,** '\_blank'**);**

$**(**'a:not([href\*=upc.edu.pe])'**).**css**(**'background-image'**,** 'url("/img/externo.gif")'**);**

**Recomendado**

**var** enlacesExternos **=** $**(**'a:not([href\*=upc.edu.pe])'**);**

enlacesExternos**.**attr**(**'target'**,** '\_blank'**);**

enlacesExternos**.**css**(**'background-image'**,** 'url("/img/externo.gif")'**);**

$**(**'a:not([href\*= upc.edu.pe])'**)**

**.**attr**(**'target'**,** '\_blank'**)**

**.**css**(**'background-image'**,** 'url("/img/externo.gif")'**)**

Los ejemplos anteriores son correctos, sin embargo por el tema del rendimiento es necesario usar lo que se recomienda en es te punto.

1. Usar **{}** en vez de **new** Object**()**.

**Incorrecto**

**var** o **=** **new** Object**();**

o**.**nombres **=** 'Oscar'**;**

o**.**apellidos **=** 'Gomez'**;**

o**.**algunaFuncion **=** **function()** **{**

console**.**log**(this.**nombres**);**

**}**

**Correcto**

**var** o **=** **{**

nombres**:** 'Oscar'**,**

apellidos**:** 'Gomez'**,**

algunaFuncion **:** **function()** **{**

console**.**log**(this.**nombres**);**

**}**

**};**

1. Usar **[]** en vez de **new** Array**().**

**Incorrecto**

**var** cursos **=** **new** Array**();**

a**[**0**]** **=** 'Programación 1'**;**

a**[**1**]** **=** 'Programación 21'**;**

**Correcto**

**var** cursos **=** **[**'Programación 1'**,** 'Programación 2'**];**

1. **CODIFICACION MVC.NET**
2. **Reglas Generales**
3. No usar variables de sesión a través del código. Usar variables y métodos de para acceder el valor almacenado en las variables de sesión.

**public** enum SessionKey

**{**

UsuarioId**,**

Nombre**,**

Apellido**,**

Codigo**,**

Email**,**

Culture**,**

Rol

**}**

**public** static class SessionHelper

**{**

#region Set & Get

**public** static **object** Get**(**HttpSessionStateBase Session**,** SessionKey Key**)**

**{**

**return** Session**[**Key**.**ToString**()];**

**}**

**public** static **object** Get**(**HttpSessionState Session**,** SessionKey Key**)**

**{**

**return** Session**[**Key**.**ToString**()];**

**}**

**public** static void Set**(this** HttpSessionStateBase Session**,** SessionKey Key**,** **object** value**)**

**{**

Session**[**Key**.**ToString**()]** **=** value**;**

**}**

**public** static void Set**(this** HttpSessionState Session**,** SessionKey Key**,** **object** value**)**

**{**

Session**[**Key**.**ToString**()]** **=** value**;**

**}**

#endregion

#region IsLoggedIn

**public** static Boolean IsLoggedIn**(this** HttpSessionState Session**)**

**{**

**return** Get**(**Session**,** SessionKey**.**Rol**)** **!=** **null;**

**}**

**public** static Boolean IsLoggedIn**(this** HttpSessionStateBase Session**)**

**{**

**return** Get**(**Session**,** SessionKey**.**Rol**)** **!=** **null;**

**}**

#endregion

#region GetUsuarioId

**public** static Int32**?** GetUsuarioId**(this** HttpSessionStateBase Session**)**

**{**

**return** **(**Int32**?)**Get**(**Session**,** SessionKey**.**UsuarioId**);**

**}**

**public** static Int32**?** GetUsuarioId**(this** HttpSessionState Session**)**

**{**

**return** **(**Int32**?)**Get**(**Session**,** SessionKey**.**UsuarioId**);**

**}**

#endregion

**}**

1. No almacenar objetos grandes en la sesión. Almacenar objetos grandes en la sesión puede consumir mucha memoria del servidor dependiendo del número de usuarios.

Session**[**"ListUsuarios"**]** **=** context**.**Usuario**.**Tolist**();**

1. No tocar el archivo Layout.cshtml, El Layout tiene los componentes necesarios para que la página web funcione correctamente. Si se necesita funcionalidad adicional, utilizar los *sections* para añadir funciones, referencias de estilos, scripts desde la vista:

**Layout.cshtml**

**@{**

**var idioma = @Session.GetCulture();**

**}**

<!DOCTYPE html>

<html lang=**"@idioma"**>

<head>

<title>**Comite de Acreditación - @ViewBag.Title**</title>

<!-- Stylesheets -->

<link rel=**"stylesheet"** href=**"@Url.Content("**~/Content/css/bootstrap.min.css")**">**

**<link rel="**stylesheet" href=**"@Url.Content("**~/Content/css/bootstrap-extend.min.css")**">**

**<link rel="**stylesheet" href=**"@Url.Content("**~/Content/css/site.min.css")**">**

**@RenderSection("**Styles", false)

<!-- End Stylesheets -->

</head>

<body>

<nav >

**...**

</nav>

<div class=**"site-menubar"**>

<div class=**"site-menubar-body"** style=**"height: 100% !important;"**>

**@{Html.RenderPartial("\_Sidebar");}**

</div>

</div>

<!-- Page -->

<div class=**"page animsition"**>

<div class=**"page-main"**>

**@if (IsSectionDefined("Breadcrumbs"))**

**{**

<ol class=**"breadcrumb"**>

<li><a href=**"@Url.Action("**Dashboard",**"Home"**)**">Home</a></li>**

**@RenderSection("**Breadcrumbs", false)

<li class=**"active"**>**@ViewBag.Title**</li>

</ol>

**}**

<div class=**"page-content"**>

**@RenderBody()**

</div>

</div>

</div>

<!-- End Page -->

<!-- Core -->

<script src=**"@Url.Content("**~/Content/vendor/jquery/jquery.min.js")**" )"**></script>

<script src=**"@Url.Content("**~/Content/vendor/bootstrap/bootstrap.min.js")**"></script>**

**<!-- End Core -->**

**@RenderSection("**Scripts", false)

</body>

</html>

**Vista**

**@model UPC.CA.Mockup.ViewModels.Report.ViewReportViewModel**

**@{**

**ViewBag.Title = "Reporte";**

**ViewBag.Icon = "bar-chart";**

**}**

**@section Styles{**

<link rel=**"stylesheet"** href=**"@Url.Content("**~/Content/css/pagina-inicio.css")**"**>

**@section Breadcrumbs{**

**}**

**<div class="**row">

<div class=**"col-md-12"**>

<div class=**"panel"**>

**...**

</div>

</div>

</div>

**@section Scripts{**

<script src=**"@Url.Content("**~/Content/js/panel.min.js")**"**></script>

<script src=**"@Url.Content("**~/Content/js/pages/view.reporte.js**"**)**"**></script>

**}**

1. No tocar el archivo **Layout.cshtml**, El Layout tiene los componentes necesarios para que la página web funcione correctamente. Si se necesita funcionalidad adicional, utilizar los *sections* para añadir dependencias, referencias de estilos, scripts desde la vista.
2. Usar reglas de validación específicas para cada elemento en un **ViewModel** utilizando **DataAnnotations** para validar los campos desde el servidor.

**public** class EmployeeViewModel

**{**

**[**Required**(**ErrorMessage**=**"El campo Empleado no puede estar vacío"**)]**

**public** String Nombre **{** get**;** set**;** **}**

**...**

**}**

1. Utilizar una declaración de **DataAnnotations** por línea.

**Correcto**

**public** class EditUsuarioViewModel

**{**

**[**Required**]**

**[**StringLength**(**11**,** MinimumLength **=** 7**)]**

**[**Display**(**Name **=** "NumeroDocumento"**,** ResourceType **=** **typeof(**Globalization**.**User**.**EditUsuario**))]**

**public** String NumeroDocumento **{** get**;** set**;** **}**

**[**Required**]**

**[**Display**(**Name **=** "Nombres"**,** ResourceType **=** **typeof(**Globalization**.**User**.**EditUsuario**))]**

**public** String Nombres **{** get**;** set**;** **}**

**[**Required**]**

**[**StringLength**(**80**)]**

**[**Display**(**Name **=** "ApellidoPaterno"**,** ResourceType **=** **typeof(**Globalization**.**User**.**EditUsuario**))]**

**public** String ApellidoPaterno **{** get**;** set**;** **}**

**}**

**Incorrecto**

**public** class EditUsuarioViewModel

**{**

**[**Required**,** StringLength**(**11**,** MinimumLength **=** 7**,** Display**(**Name **=** "NumeroDocumento"**,** ResourceType **=** **typeof(**Globalization**.**User**.**EditUsuario**))]**

**public** String NumeroDocumento **{** get**;** set**;** **}**

**[**Required**,** Display**(**Name **=** "Nombres"**,** ResourceType **=** **typeof(**Globalization**.**User**.**EditUsuario**))]**

**public** String Nombres **{** get**;** set**;** **}**

**[**Required**,** StringLength**(**80**),** Display**(**Name **=** "ApellidoPaterno"**,** ResourceType **=** **typeof(**Globalization**.**User**.**EditUsuario**))]**

**public** String ApellidoPaterno **{** get**;** set**;** **}**

**}**

1. Utilizar cache en componentes de uso reiterado, data compartida, que no requiera autorización y de pocos cambios. De esta manera el ancho de banda de red entre el cliente y el servidor también se reduce.

**public** class HomeController **:** Controller

**{**

**[**OutputCache**(**Duration**=**3600**,** VaryByParam**=**"none"**)]**

**public** ActionResult Index**()**

**{**

...

**}**

**}**

1. Crear un **ViewModel** por cada Vista. Evita usar **ViewBag** para enviar datos del controlador a la vista.

**Correcto**

**public** ActionResult Login**()**

**{**

var viewModel **=** **new** LoginViewModel**();**

**return** View**(**viewModel**);**

**}**

**Incorrecto**

**public** ActionResult Login**()**

**{**

ViewBag**.**Usuario **=** String.Empty**;**

ViewBag**.**Password **=** String.Empty**;**

**return** View**();**

**}**

1. El **Model** de una vista siempre será instanciado a partir de un **ViewModel**.

**@model UPC.CA.Mockup.ViewModels.Report.ViewReportViewModel**

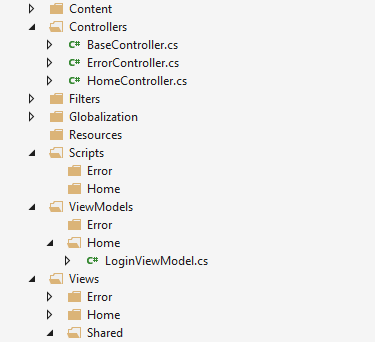
**@{**

**ViewBag.Title = "Reporte";**

**ViewBag.Icon = "bar-chart";**

**}**

1. Usar **ViewModel** para las vistas pasar datos a las vistas y jamás expongas las Entidades.
2. Guardar la estructura de carpetas basado en la estructura del controlador, dentro de los **Views**, **ViewModel**, y **Scripts**.



1. Usar notación Pascal parámetros de las Clases con la siguiente estructura dentro de un **ViewModel**.

public class LoginViewModel

{

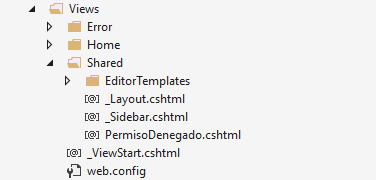
public String Usuario { get; set; }

public String Password { get; set; }

...

}

1. Si necesitas un dato extra para la vista añádelo al **ViewModel**.
2. No acceder a la capa de datos desde una vista.
3. Conserva las vistas limpias de código que no sea **HTML**.
4. Si una vista se repite, se podrá convertir en una vista parcial.
5. Nombrar las vistas parciales y compartidas con un subguión (\_) de prefijo.



1. Si una vista puede ser reutilizada en otros **Controller**, conviértela en una vista compartida y nómbra la acción con un subguión (\_) de prefijo dentro del **Controller**.

**public** class ErrorController **:** BaseController

**{**

// GET: Error

**public** ActionResult NotFound**()**

**{**

**return** View**();**

**}**

**public** ActionResult InternalServerError**()**

**{**

**return** View**();**

**}**

**public** ActionResult \_PermisoInsuficiente**()**

**{**

**return** View**();**

**}**

**}**

**BIBLIOGRAFIA**

https://msdn.microsoft.com/en-us/library/ms229042%28v=vs.100%29.aspx