**VII. Convenciones o estándares de Programación**

Los estándares de programación o codificación son las reglas para normalizar la nomenclatura de los códigos fuentes que se utilizaran en el proceso de desarrollo del sistema en defecto, dichas nomenclaturas afectaran tanto al código fuente en si tal como páginas web, lógica de negocios, diseño de base de datos, etc.

**Metodología de Programación**

Debido al corto tiempo del cual disponemos, basada en planificaciones ya realizadas con anterioridad por el equipo de planificación que serán realizadas por los equipos asignados, pudiendo estos estar compuestos de forma unipersonal mediante la asignación de tareas.

Inicialmente se compondrá de 3 integrantes, los cuales estarán a cargo de la codificación, uno de los integrantes responderá por los 2 integrantes tanto en testing documentación y distribución de tares.

**Herramientas colaborativas:** issue/tracker code google,SVN Code google, Gantt, Microsoft Web Developer Express, Microsoft SQL Server Management Studio Express.

**Datos del Sistema**

**Base de Datos:** MySQL

**Lenguaje de Programación:** Ruby on Rails.

**Editor:** Sublime Text 2 ó Aptana Studio 3.

**Bases de Datos:**

\*Los nombres las Bases de Datos se escribirán en minúsculas.

\* Los nombres de todos los objetos de la Base de Datos se escribirá en plural.

\* La documentación del programa será en español

**Nomenclatura de las Tablas**

Los nombres de las tablas estarán escritas en singular o en plural compuestas por cadenas separadas entre sí por guiones bajos donde el primer carácter separados por los guiones bajos o solamente un nombre cuyo carácter empiece en minúsculas.

***Ejemplo:***

clientes

clientes\_detalle

**Campos definidos dentro de las tablas**

El formato a utilizar en los campos  definidos dentro de las tablas serán igual a la  nomenclatura de las tablas detalla en el punto anterior.

**Código de Aplicaciones**

**Nombre del Proyecto, archivos fuentes y estructura del proyecto**

Las aplicaciones Ruby on Rails generan automáticamente directorios, el nombre del proyecto estará relacionado con el modulo a desarrollar y estará todo en minúsculas y en caso de estar compuesto estarán separadas las palabras por el guion bajo.

Al desarrollar una aplicación Ruby on Rails por defecto se generan las siguientes carpetas:

*<nombre\_proyecto>/*

La carpeta principal que contiene todas las subcarpetas y archivos del proyecto. Tiene el mismo nombre del proyecto.  
*<nombre\_proyecto>/app*

La carpeta que contiene subcarpetas en la que mayormente trabajaremos. Contiene el código fuente principal del proyecto.  
*<nombre\_proyecto>/app/controller*

Esta carpeta contiene todos los archivos controladores, los nombres deben de comenzar siempre con la palabra controller.  
*<nombre\_proyecto>/app/helpers*

Esta carpeta los archivos de ayuda para las clases controladores.  
*<nombre\_proyecto>/app/models*

Esta carpeta contiene los modelos, que deben de ir relacionados con las tablas de base de datos. Si existe alguna tabla llamada usuario, debe existir también un modelo llamado usuario con sus mismos atributos llamados de la misma forma que en la base de datos.  
*<nombre\_proyecto>/app/views*

Esta carpeta contiene las vistas del proyecto, todos los archivos html.erb, formularios y demás interfaces del sistema. Dentro de esta carpeta debemos de crear otras por cada modelo creado, con el mismo nombre del modelo que contendrá los métodos del controlador como ser show, index, new, edit.  
*<nombre\_proyecto>/app/views/layouts*

Esta carpeta contiene los archivos que serán parte de todas las vistas.  
*<nombre\_proyecto>/config*

La carpeta contendrá archivos de configuración que se cargarán al levantar la aplicación. Varias gemas utilizadas deberán ser configuradas aquí.  
*<nombre\_proyecto>/db:* para archivos relacionados a base de datos, ya sea para su creación, edición o delete.  
*<nombre\_proyecto>/doc:* para la documentación.  
*<nombre\_proyecto>/lib:* para librerías adicionales  
*<nombre\_proyecto>/log:* contiene el log del proyecto.  
*<nombre\_proyecto>/public:* contiene los archivos públicos del proyecto.  
*<nombre\_proyecto>/script:* contiene los scripts del proyecto.  
*<nombre\_proyecto>/test:* aquí realizaremos los test de Ruby del proyecto  
*<nombre\_proyecto>/tmp:* archivos temporales  
*<nombre\_proyecto>/vendor*

En este directorio se almacenan bibliotecas provistas por terceras partes (como por ejemplo bibliotecas de seguridad o utilitarios de base de datos más allá de la distribución básica de *Rails*).  
*<nombre\_proyecto>/README*

Este archivo contiene detalles básicos sobre *Rails Application* y la descripción de la estructura de archivos descripta arriba.  
*<nombre\_proyecto>/Rakefile*

Este archivo es similar al *Makefile* de *Unix*, y sirve para construir, empaquetar y probar el codigo *Rails*. Este archivo es usado por el utilitario *rake* provisto con la instalación *Ruby.*

**Ubicación de los archivos de código fuente**

El código fuente principal se dividirá según lo establecido por la forma de programación del MVC, diviendo todo lo relacionado con las clases, ya sean los atributos, las validaciones y métodos propios de cada clase estarán bajo una carpeta llamada model en el proyecto, cada clase deberá de tener la extensión de .rb y el nombre de la clase deberá ser la misma utilizada en la tabla de base de datos.

Todo lo que sea métodos para comunicar la vista (la página html) con la clase (con el modelo) estará bajo una carpeta llamada controller, que todos los nombres de los archivos ahí dentro deberán empezar con la palabra controller y tendrá la extensión de .rb así como en el modelo.

Todo lo relacionado a páginas html estará bajo la carpeta view y tendrá la extensión .html.erb y se dividirán básicamente en show (para mostrar datos), edit(para editar datos), new(para agregar datos), index(para listar datos), y otras en casos especiales. Cada modelo tendrá todos estos archivos bajo su propia carpeta en la carpeta view.

*Por ejemplo:*

Si tengo un modelo llamado usuarios.rb

Dentro de la carpeta view debo de tener la carpeta usuarios y dentro de ella tendré los archivos show.html.erb, index.html.erb, edit.html.erb, new.html.erb…

model/usuarios.rb

controller/controller\_usuarios.rb

view/usuarios/new.html.erb

view/usuarios/show.html.erb

……

**Nombre de objetos**

Dentro de la aplicación Web, vamos a tener distintos tipos de objetos, los nombres de los mismos estarán en minúscula y deberán expresar la función a desarrollar por el objeto. Estarán completamente en minúscula y en no podrán ser compuestos.

**Clases**

Los nombres de las clases deben empezar en mayúscula y si posee un nombre compuesto se utilizará la técnica de la notación Camel empezando siempre en mayúscula. El nombre del archivo deberá estar en minúscula y si es un nombre con palabras compuestas estás deberán estar separadas por el guion bajo, siguiendo con la convención de Ruby.

Por ejemplo:

Si queremos crear una clase para gestionar los usuarios, está deberá llamarse Usuarios y el archivo se guardará como usuarios.rb

Al crear un objeto de la clase Usuarios esté podrá llamarse usuario, en caso de ser compuesto usuario\_nombre.

**Métodos**

Los nombres de los métodos deben de expresar la función del mismo. En caso de llevar parámetros, éstos deberán estar dentro de un paréntesis separado por comas.

def <nombre\_metodo>

Los nombre de los métodos deben de estar en minúsculas y en caso de ser compuestos, cada palabra deberá estar separado por el guion bajo.

**Constantes**

Los nombres de las constantes globales y locales deberán expresar la intención del mismo, en caso de ser compuestos estarán separados por el guion medio.

Los nombres de constantes siempre se escribirán en MAYUSCULA

Por ejemplo DIRECCION-LOCAL

Como el nombre de la constante lo va a definir el consultor, se debe de ser lo más descriptivo posible, dejando a entender el valor del mismo. Si el nombre de la constante esta compuesta de varias palabras, debemos de digitar las palabras unidas como nombre de la constante, las palabras deben estar en mayúscula separadas por el guion medio.

**Variables**

Las variables dependiendo de su alcance se dividen en globales y locales, el nombre de la variable consta de un prefijo seguido del nombre de la variable definido por el consultor.

El nombre debe de tener la siguiente sintaxis:

<ubicación><tipo de dato>\_<nombre de la variable>

El tipo de dato puede omitirse en caso de ser una cadena o un tipo numérico.

Los prefijos son los siguientes:

Alcance:

Global (g)

Local (l) , todas las variables locales pueden omitir el uso de este prefijo

***Tipo de Dato***

|  |  |
| --- | --- |
| Nombre | Prefijo |
| String | S |
| Boolean | B |
| Fixnum | N |
| Integer | I |
| float | L |
| List | Lst |
| Array | Arr |
| Objeto | Obj |
| DateTime | Dt |
| Date | Dt |
| Time | Dt |

…

Como el nombre de la variable lo va a definir el consultor, debe ser lo más claro posible, dejando a entender el valor del mismo. Si el nombre de la variable consta de varias palabras, debemos de digitar las palabras separadas por el guion bajo como nombre de la variable.

*Por ejemplo:*

Si vamos a necesitar una variable que va a contener un primer nombre (dos palabras) de una persona, debe ser como el siguiente ejemplo:

ls\_nombre\_empleado

Si necesitamos crear una variable que va a contener un estado (una palabra), debe ser como el siguiente ejemplo:

ls\_estado o s\_estado, como es um string puede solo llamarse estado.

Variables de contadores

Se debe utilizar i, j, k, l, m, n para los nombres para las variables de contadores utilizados.

Posición de variables dentro del Código Fuente

Todas las variables sin distinción deben quedar declaradas en la cabecera o header del método, función o evento. A pesar que el código fuente permita otra acción se requiere que todas queden declaradas en la posición indicada.

Por ejemplo:

def XYZ

ln\_edad = 1

nombre = “Pedro”

array\_usuarios = []

Un mal ejemplo

def XYZ

ln\_edad = 1

nombre = “Pedro”

if llamarDatos()

#cualquier\_cosa

end

array\_usuarios = []

Controles Visuales

Los controles visuales deben poseer un nombre estándar, el nombre debe ser el mismo con el cual fue nombrado la clase, el mismo nombre del atributo declarado en el modelo del objeto deberá de ser usado en la vista, es decir, en las páginas html.erb ya sea para mostrar el atributo, para editar, guardar o listar.

**Documentación interna**

**Métodos**

Para los métodos debemos de seguir los siguientes estándares. Antes de la declaración de cada método debemos de colocar un encabezado.

**Por ejemplo:**

#Explicar para qué sirve el método creado

def método

end

Este encabezado debe ser agregado una línea encima del método, debe explicar cuál es el propósito del método creado.

Dentro de cada método opcionalmente po<nombre\_proyecto>s poner líneas con comentarios si es que escribimos algún código con dificultad de entenderse. Éstas no deberán de superar una línea como máximo.

**Código fuente comentado**

Se entiende como código fuente comentado aquellas líneas de código fuente que fueron puestas entre comentarios por fines de reemplazo de código. Este código comentado no tiene funcionalidad alguna dentro del sistema por lo tanto debe de ser eliminado por completo del mismo, solo en casos excepcionales.

**Por ejemplo:**

def cargarSegundoCampo

#condicion = false

#if condición

return CAMPO

end

Las líneas anteriores comentadas no deben de permanecer en el código fuente estable que se usará para realizar el sistema.

Esta virtud de programación será altamente evaluada durante el proceso de auditoría.

**Tabulación**

Se respetará lo establecido convencionalmente por Ruby, la tabulación será de 2(dos) espacios entre cada bloque.

**Por ejemplo:**

Class MyClass

Def initialize

@my\_class = “Mi clase”

end

**Asignación de Variables y el uso de signos.**

Para inicializar cualquier objeto en el sistema seguiremos también un estándar, pondremos el nombre del objeto y/o variable, un espacio, el signo = seguido de un espacio y el valor asignado. Siempre que escribamos un signo ya sea el signo + - \* = dejaremos espacios antes y espacios después, no es necesario estos espacios entre signos.

Por ejemplo:

array\_usuarios = []

suma += 1

if a == b

# Formatos

**Formato de Fecha**

Los formatos de fechas deben ser dd/MM/yyyy.

**Formato de Mensajes**

Mensajes de validación de campos requeridos

Los mensajes de validación de campos requeridos deben ir al costado de los campos y deben ser de color rojo. Ejemplo:

Ingrese Descripción

Mensajes de validación de fechas

Los mensajes de validación de fechas deben ir al costado de los campos y deben ser de color rojo. Ejemplo:

Fecha (dd/mm/yyyy) entre: 1/1/2000-31/12/2078

Mensajes de validación de búsqueda de información

Los mensajes de validación de búsqueda de información deben ir en la parte inferior de la pantalla y deben ser “No se encuentran registros”.

**Diseño de Interfaz y lugar de Programación Web**

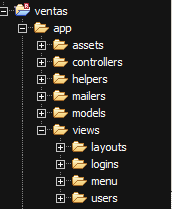
**Interfaz Web**

Las interfaces web tienen ciertas limitaciones en las funcionalidades que se ofrecen al usuario. Hay funcionalidades comunes en las aplicaciones de escritorio como dibujar en la pantalla o arrastrar y soltar que no están soportadas por las tecnologías web estándar. Los desarrolladores web generalmente utilizan lenguajes interpretados (scripts) en el lado del cliente para añadir más funcionalidades, especialmente para ofrecer una experiencia interactiva que no requiera recargar la página cada vez (lo que suele resultar molesto a los usuarios). Recientemente se han desarrollado tecnologías para coordinar estos lenguajes con las tecnologías en el lado del servidor. Como ejemplo, AJAX es una técnica de desarrollo web que usa una combinación de varias tecnologías.

**Lugar de Programación Web**

Dentro del marco de directorios que utilizar ruby on rails contamos un lugar donde se alojan las interfaces web de usuario

Dicha interfaces se encuentaran dentro del directorio app/views como se muestra en el siguiente ejemplo



Como vemos dentro del app se encuentra la carpeta views que contienen todas las plantillas que se le mostraran al usuario para que interactúe con la aplicación, dichas plantillas están asociado un nombre de carpeta que esta pluralizado

Dentro del mismo vemos un carpeta llamada layouts contiene las plantillas para los esquemas a utilizar con las vistas. Estas plantillas modelan los métodos header/footer comunes de empaquetado de vistas. Dentro del mismo de define el siguiente esquema a nivel de código fuente.

application.html.erb

<!DOCTYPE html>

<html>

<head>

<title></title>

<%= stylesheet\_link\_tag "application", :media => "all" %>

<%= javascript\_include\_tag "application" %>

<%= csrf\_meta\_tags %>

</head>

<body>

<%= yield %>

</body>

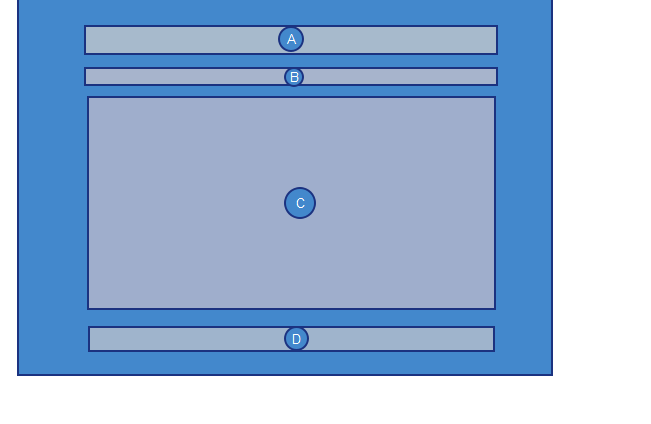
</html>

Dentro del body se encuentra el <%= yield %> que es donde se escribe código ruby

Todas las páginas web realizadas estarán atadas a seguir la estructura de diseño de página que está dentro de application.html.erb.

**Estructura del Diseño Web**

La gran mayoría de las páginas web tienen divisiones o secciones similares. Lo usuales que, en la parte superior de la página se encuentre un encabezado, un nivel más abajo esta de barra de navegación seguido de un cuerpo principal que suele estar flanqueado por una columna lateral , seguidas al final de la página por un pie de página. La imagen siguiente es un esquema típico de una página web.



Las secciones serán las siguientes:

1. Barra de título (Header)
2. Barra de navegación(Menu)
3. Área de trabajo(Body)
4. Pie de página (Footer)

**Estilos CSS a las Páginas Web**

Las hojas de estilo CSS nos permiten separar el contenido de la página de la presentación de la misma. Asimismo, la capacidad de añadir más de un estilo a la página web nos permite:

* Cambiar su diseño visual sin cambiar su contenido
* Mantener el código HTML y CSS limpio y claro
* Añadir y combinar elementos de diseño por referencia (sin incluirlos directamente en el archivo HTML)

En ruby on rails existe una gran cantidad de gemas que son pequeñas librerías que facilitan el trabajo del programador, utilizaremos una gema en especial para el diseño de CSS de las páginas web llamada twitter-bootstrap.

**Contenidos de pantallas (Paginas Web)**

El sistema tendrá varias pantallas, diseñadas en el análisis de diseño usado en la metodóloga RUP, entre ellas mencionamos las principales a continuación:

Pantalla de Ingreso: Pantalla de autenticación de usuarios, aquí se solicita la identificación de usuario y clave de acceso. Posteriormente el sistema verifica la validez de información, de lo contrario se deniega el acceso.

Pantalla de Bienvenida e Información al sistema: Esta pantalla es la siguiente que verá un usuario, ya luego de loguearse. La página tendrá una breve información de la empresa donde el usuario podrá acceder a un menú de navegación e ingresar a ciertos ítems del mismo de acuerdo al rol que tenga dicho usuario .