**Sistema de ECOMMERCE MIJOSTORE**

**Documento de Estándares de Programación**

**Versión 1.0**

**Historia de Revisión**

| **Historial de revisiones** | | | | | |
| --- | --- | --- | --- | --- | --- |
| **Ítem** | **Fecha** | **Versión** | **Descripción** | **Equipo** | |
| 1 |  | 1.0 | Versión Final. |  | |

**Tabla de Contenidos**

[**1. OBJETIVOS 4**](#_heading=h.asxqpbktzgls)

[**2. DECLARACIÓN DE VARIABLES 5**](#_heading=h.r7i190edobnn)

[2.1 Descripción de la Variable. 6](#_heading=h.9kia0rcbseis)

[**3. Definición de Controles 6**](#_heading=h.acwa9qywrq59)

[3.1 Tipo de datos 7](#_heading=h.65z2963ztxza)

[3.2 Prefijo para el Control 7](#_heading=h.n0qjk1ixh2nb)

[3.3 Nombre descriptivo del Control 7](#_heading=h.mmj89exg8thk)

[3.4 Declaración de variables y atributos 8](#_heading=h.axot5bpxpj8m)

[3.5 Declaración de clases 26](#_heading=h.i7131yuqx1s8)

[3.6 Declaración de métodos 31](#_heading=h.y1ag4rcmkzj1)

[3.7 Declaración de funciones 37](#_heading=h.lnxbz9)

[3.8 Control de versiones de código fuente 43](#_heading=h.cl42s5ekyyyq)

[**4. Clases. 43**](#_heading=h.44sinio)

[**5. Métodos y Funciones definidos por el Usuario. 44**](#_heading=h.ehgwqavtdls3)

[**6. Beneficios 45**](#_heading=h.3ikyy2v8r484)

[**7. Conclusión 45**](#_heading=h.6it2m2muxp6a)

**Estándares de Programación**

# OBJETIVOS

* Establecer un conjunto integral de estándares de programación que abarque desde la estructura del código hasta las prácticas de documentación y control de versiones, con el fin de garantizar la consistencia y la calidad en todos los aspectos del desarrollo de software.
* Promover la modularidad y la reutilización de código a través de la definición de interfaces claras y la separación de preocupaciones, con el fin de facilitar el mantenimiento, la escalabilidad y la extensibilidad de los sistemas de software.
* Fomentar la adopción de prácticas de desarrollo ágil, como la entrega continua y la automatización de pruebas, para acelerar el ciclo de desarrollo, mejorar la calidad del software y adaptarse más fácilmente a los cambios en los requisitos del proyecto.
* Facilitar la colaboración y el intercambio de conocimientos entre los miembros del equipo de desarrollo mediante la creación de una documentación exhaustiva de los estándares de programación, la organización de sesiones de formación y la celebración regular de reuniones para revisar y actualizar los estándares según sea necesario.

# DECLARACIÓN DE VARIABLES

Se propone que la declaración de las variables, se ajusten al motivo para la que se requieran. El mnemotécnico definido se establece tomando en consideración principalmente lo siguiente:

* La longitud debe ser lo más recomendable posible. No debe ser tan grande de tal forma que el programador tenga la facilidad de manejo sobre la variable y ni tan corta que no pueda describirse claramente. Para el caso establecemos una longitud máxima de variable de 16 caracteres.
* Alcance de la variable

A medida que aumenta el tamaño del proyecto, también aumenta la utilidad de reconocer rápidamente el alcance de las variables. Esto se consigue al escribir un prefijo de alcance de una letra delante del tipo de prefijo propio, sin aumentar demasiado la longitud del nombre de las variables.

| **Alcance** | **Prefijo** | **Ejemplo** | **Ejemplo** |
| --- | --- | --- | --- |
| Global | G | gClientMongoDB | Variables globales accesibles en toda la aplicación. |
| Nivel de la clase | *M* | mCollectionUsuarios | Variables usadas dentro de una clase. |
| Local del procedimiento / método | Ninguno | client, collection | Variables declaradas dentro de un método o función. |
| Público | P | pResultadoInsercion | Variables públicas expuestas fuera de la clase. |
| Privado | Pr | prClaveHash, prDireccionEnvio | Variables privadas utilizadas solo dentro de la clase. |

* El tipo de dato al que pertenece la variable.

Por lo tanto la estructura de la variable es como sigue:

| **Estructura** | **Descripción de la Variable** |
| --- | --- |
| LONGITUD. MAX. | 🡨 1 🡪🡨 16 🡪 |
| FORMATO | *Minúscula la primera parte y luego la segunda con Mayúsculas* |
| EJEMPLO | clienteMongo |

## Descripción de la Variable.

Nombre que se le asignará a la variable para que se le identifique y deberá de estar asociada al motivo para la cual se le declara.

| **Nombre de la Variable** | **Descripción** | **Motivo para Declaración** |
| --- | --- | --- |
| gClientMongoDB | Cliente global para la conexión a MongoDB. | Se utiliza para gestionar la conexión en toda la aplicación. |
| mCollectionUsuarios | Colección de usuarios usada a nivel de la clase. | Maneja operaciones CRUD para la colección de usuarios. |
| prClaveHash | Clave encriptada del usuario. | Protege la seguridad de la contraseña antes de almacenarla. |
| prDireccionEnvio | Dirección de envío proporcionada por el usuario. | Se utiliza para procesar los pedidos y asignar la dirección correcta. |
| pResultadoInsercion | Resultado de la inserción pública. | Informa si una operación fue exitosa al cliente que la solicitó. |
| documentoPedido | Documento que contiene los detalles del pedido. | Maneja la estructura de datos del pedido antes de ser insertado. |
| categoriaId | ID de la categoría asociada al producto. | Identifica a qué categoría pertenece un producto específico. |
| idProducto | Identificador único del producto. | Usado para referenciar un producto en operaciones de base de datos. |
| estadoPedido | Estado actual del pedido, como "pendiente" o "completado". | Indica el progreso del pedido para actualizaciones y notificaciones. |
| fechaRegistro | Fecha en que se registró un usuario o pedido. | Permite realizar análisis y ordenamientos basados en la fecha de registro. |
| tipoPromocion | Tipo de promoción activa, como "Black Friday". | Identifica el tipo de promoción que se aplicará a un producto. |
| prNuevosDatos | Datos nuevos para actualizar un documento en la base de datos. | Usado para actualizaciones parciales de un documento existente. |
| client | Cliente MongoDB utilizado dentro de un método específico. | Maneja la conexión dentro del contexto del método. |

# Definición de Controles

## Tipo de datos

| **Tipo de variable** | **Mnemónico** | **Descripción** |
| --- | --- | --- |
| Int | Ent | Representa números enteros. |
| String | Str | Representa cadenas de texto |
| LocalDate | LocalDate | Representa una fecha sin zona horaria local. |
| String | da | Cadena de caracteres |
| Date | Dat | Representa una fecha con zona horaria. |
| Enum | Est | Representa un enum de un estado o condición de algo. |
| Enum | Gen | Representa un enum de la identidad de género de una persona. |
| Time | Tim | Representa una hora específica del día. |

## Prefijo para el Control

El prefijo del control se crea con tres letras que reducen el nombre de las herramientas utilizadas. Por ejemplo, para "formulario", el prefijo sería "form".

## Nombre descriptivo del Control

Formado por la descripción de la función que lleva a cabo el control, esta debe ser descrita en forma específica y clara.

| **Tipo de control** | **Prefijo** | **Ejemplo** |
| --- | --- | --- |
| Label | lbl | lblNombre |
| TextBox | txt | txtNombre |
| Button | btn | btnCargar |
| Select | slct | selectDistrito |
| href | href | href=producto/ver\_promocion&id |

## Declaración de variables y atributos

| **Título** | **Descripción** |
| --- | --- |
| **Sintaxis** | private $id; |
| **Descripción** | Variable privada que almacena el identificador único para cada instancia de la clase Caracteristica. |
| **Observaciones** | En la declaración de variables o atributos no se deberá utilizar caracteres como:   * Letra Ñ o ñ. * Caracteres especiales ¡, ^, #, $, %, &, /, (, ), ¿, ‘, +, -, \*, {, }, [, ]. * Caracteres tildados: á, é, í, ó, ú. |
| **Ejemplo** | if($identidad && is\_object($identidad)) |

## Declaración de clases

| **Título** | **Descripción** |
| --- | --- |
| **Sintaxis** | class Caracteristica { } |
| **Descripción** | Caracteristica es una clase que representa las características específicas de un objeto. Puede incluir atributos como ID, nombre, valor, y métodos para manipular y acceder a esos atributos. |
| **Observaciones** | En la declaración de clases no se deberá utilizar caracteres como:   * Letra Ñ o ñ. * Caracteres especiales ¡, ^, #, $, %, &, /, (, ), ¿, ‘, +, -, \*, {, }, [, ]. * Caracteres tildados: á, é, í, ó, ú. |
| **Ejemplo** | public class ClsNAnimal |

## Declaración de métodos

| **Título** | **Descripción** |
| --- | --- |
| **Sintaxis** | public boolean guardarMongoDB() |
| **Descripción** | El método guardarMongoDB() define una función pública que guarda un documento de usuario en una base de datos MongoDB. Este método encripta la contraseña antes de almacenarla y luego inserta los datos del usuario en la colección usuarios de la base de datos MiTiendaDB |
| **Observaciones** | En la declaración de métodos no se deberá utilizar caracteres como:   * Letra Ñ o ñ. * Caracteres especiales ¡, ^, #, $, %, &, /, (, ), ¿, ‘, +, -, \*, {, }, [, ], \_. * Caracteres tildados: á, é, í, ó, ú. |
| **Ejemplo** | php <br> $objNal->guardarMongoDB(); <br>  Este método se utiliza para guardar un usuario en MongoDB, asegurando que la contraseña se almacene de manera segura. El método devuelve true si la inserción fue exitosa y false en caso de error. |

## Declaración de funciones

| **Título** | **Descripción** |
| --- | --- |
| **Sintaxis** | public function cerrarSesion() |
| **Descripción** | El método cerrarSesion() es una función pública que se utiliza para cerrar la sesión del usuario. Este método verifica si existen las variables de sesión identity y admin, y si están definidas, las elimina. Luego, redirige al usuario a la página principal. |
| **Observaciones** | En la declaración de objetos no se deberá utilizar caracteres como:   * Letra Ñ o ñ. * Caracteres especiales ¡, ^, #, $, %, &, /, (, ), ¿, ‘, +, -, \*, {, }, [, ], \_. * Caracteres tildados: á, é, í, ó, ú. |
| **Ejemplo** | cerrarSesion();  Este método se utiliza para cerrar la sesión de un usuario y redirigirlo al sitio principal. Elimina las variables de sesión relacionadas con la identidad y el administrador. |

## Control de versiones de código fuente

Cada modificación realizada será guardada de la forma:

| **Título** | **Descripción** |
| --- | --- |
| **Formato** | git commit -m “[Nombre del FRM] - [Descripción del avance]” |
| **Descripción** | Se generarán commits de acuerdo al avance del form con una breve descripción de las modificaciones. Por ejemplo: Login - Avance de interfaz |

# Clases.

**Entidad:**

* class Caracteristica
* class Categoria
* class Imagen
* class Pedido
* class Producto
* class Promocion
* class TipoPromocion
* class Usuario

# Métodos y Funciones definidos por el Usuario.

**Métodos:**

* index()
* agregar()
* eliminar()
* quitar()
* aumentar()
* reducir()
* crear()
* guardar()
* mostrar()
* mostrarproductosxpromocion()
* eliminar()
* editar()
* pedido()
* agregar()
* confirmado()
* descargar()
* misPedidos()
* detallesPedido()
* gestionar()
* actualizarEstado()
* index()
* setEstaLogueado($estaLogueado)
* getEstaLogueado()
* guardar()
* iniciarSesion()
* perfil()
* editarPerfil()
* cerrarSesion()
* actualizarPerfil()

**Funciones:**

* obtenerCaracteristicasProducto($producto\_id)
* guardar()
* actualizar()
* eliminar()
* obtenerCategorias()
* obtenerCategoria()
* guardar()
* eliminar()
* obtenerCategoriaActual()
* actualizar()
* obtenerImagenesProducto($producto\_id)
* guardar()
* actualizar()
* eliminar()
* guardar\_imagen($producto\_id, $ruta\_completa)
* obtener\_imagen\_por\_id($id)
* eliminar\_imagen($id)
* actualizar\_imagen($imagen\_id, $producto\_id, $imagen\_url)
* obtenerProductos()
* guardar()
* obtenerProductoActual()
* guardarProductosPedido()
* obtenerUltimoPedido()
* obtenerTodosPorUsuario()
* obtenerPedidoEspecifico()
* obtenerUltimoPorUsuario()
* obtenerProductosPorPedido()
* obtenerUnidades()
* actualizar()
* guardar()
* actualizar()
* desactivar()
* eliminar()
* obtenerPromociones()
* obtenerPromocionesProducto()
* obtenerPorId()

# Beneficios

* Establece reglas y convenciones que garantizan que el código sea uniforme y fácilmente comprensible por todos los miembros del equipo.
* Los estándares promueven buenas prácticas de codificación, lo que puede llevar a la reducción de errores y a un código más robusto y mantenible.
* Al seguir estándares comunes, los desarrolladores pueden trabajar de manera más eficiente en equipo, entender el código de los demás y colaborar con mayor fluidez.
* Evitar debates innecesarios sobre la estructura y el estilo del código, permitiendo a los desarrolladores concentrarse en tareas más importantes.
* El código estandarizado es más fácil de integrar con otros sistemas y de escalar a medida que el proyecto crece.

# Conclusión

La documentación de estándares de programación es fundamental para asegurar la consistencia y calidad del código en un proyecto de desarrollo de software. Al establecer reglas y convenciones claras, promueve la colaboración efectiva, mejora la mantenibilidad del código y contribuye a la eficiencia y éxito del equipo de desarrollo. Es una inversión importante que puede proporcionar beneficios a largo plazo en términos de productividad y calidad del producto final.