**Sistema de PETCARE**

**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

#### Longitud de la Variable

Se sigue manteniendo la longitud recomendada de **16 caracteres máximo** para las variables, con el fin de equilibrar claridad y facilidad de uso, aunque con algunas excepciones para las claves Redis (que pueden ser más largas si es necesario para reflejar jerarquías o identificadores más específicos).

#### Alcance de la Variable

Los alcances siguen los mismos principios, pero adaptados a las estructuras de Redis:

| **Alcance** | **Prefijo** | **Ejemplo** |
| --- | --- | --- |
| Global | G | gHshUsuarios |
| Nivel de la clase | *M* | mZsetRanking |
| Local del procedimiento / método | Ninguno | lstProductos |
| Público | P | pKeySesion |
| Privado | Pr | prSetProductosFavoritos |

* 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 | numCuenta |

## Descripción de la Variable.

Las claves en Redis deben ser descriptivas y reflejar el propósito de la variable de manera clara.

**gStrIdCuenta:12345 -> "1001"**

# Definición de Controles

## Tipo de datos

| **Tipo de variable** | **Mnemónico** | **Descripción** | **Ejemplo en Redis** |
| --- | --- | --- | --- |
| Int | Ent | Representa números enteros. | gStrCuentaBalance:1001 -> 500 |
| String | Str | Representa cadenas de texto | gStrNombreUsuario:1001 -> "Juan Pérez" |
| LocalDate | LocalDate | Representa una fecha sin zona horaria local. | gStrFechaRegistro:1001 -> "2024-12-01" |
| String | da | Cadena de caracteres | gStrDireccionUsuario:1001 -> "Av. Principal 123" |
| Date | Dat | Representa una fecha con zona horaria. | gStrFechaNacimiento:1001 -> "1990-05-10T00:00:00Z" |
| Estado | Est | Representa un enum de un estado o condición de algo. | gStrEstadoCuenta:1001 -> "Activo" |
| Género | Gen | Representa un enum de la identidad de género de una persona. | gStrGeneroUsuario:1001 -> "Masculino" |
| Time | Tim | Representa una hora específica del día. | gStrHoraUltimaConexion:1001 -> "15:30:00" |

## 

## Prefijo para el Control

El prefijo en Redis puede seguir una convención similar a la de otros lenguajes o tecnologías, donde las claves representan elementos específicos y se abrevia el nombre del tipo de control. Cada prefijo tendrá 3 letras que representan la función del control.

## 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 |
| Tabla | tb | tbNombre |
| ComboBox | cmb | cmbDistritos |

## 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** | HSET caracteristica:<id> nombre <nombre> valor <valor> |
| **Descripción** | En Redis, las características de un objeto se almacenan como un hash, donde la clave es única para cada objeto y los campos son los atributos. Por ejemplo, un objeto Caracteristica puede tener atributos como id, nombre y valor, que se guardan como campos dentro del hash.  Al usar la instrucción HSET, se puede almacenar múltiples atributos de un objeto bajo una clave que representa a ese objeto específico.  La información se puede recuperar usando HGET o HGETALL para obtener uno o todos los campos del hash. |
| **Observaciones** | * - En la declaración de las estructuras o claves en Redis no se debe utilizar caracteres como: * Letra Ñ o ñ. * Caracteres especiales como ¡, ^, #, $, %, &, /, (, ), ¿, ‘, +, -, \*, {, }, [, ]. * - Redis no utiliza clases como en lenguajes de programación tradicionales. En su lugar, se utiliza el modelo de datos clave-valor, lo que significa que cada atributo de un objeto debe ser almacenado como un campo dentro de una clave de tipo hash. |
| **Ejemplo** | php <br> HSET caracteristica:1 nombre "Color" valor "Rojo"; <br>  Este ejemplo muestra cómo almacenar una característica en Redis, donde caracteristica:1 es la clave del objeto y nombre y valor son los campos que contienen información sobre la característica. La clave y los campos permiten una recuperación eficiente y accesible de los datos. |

## Declaración de métodos

| **Título** | **Descripción** |
| --- | --- |
| **Sintaxis** | public boolean guardarRedis() |
| **Descripción** | El método guardarRedis() toma los datos de un usuario como parámetros (nombre, correo y contraseña) y los guarda en Redis usando las claves adecuadas. La contraseña es encriptada antes de ser almacenada. Los datos se guardan en Redis como claves individuales para cada atributo del usuario, facilitando el acceso eficiente. Este método realiza las siguientes operaciones:  1. **Encriptar la contraseña**: Antes de guardar la contraseña en Redis, esta se encripta para garantizar la seguridad.  2. **Guardar la información en Redis**: Los atributos como el nombre, correo y contraseña se almacenan en claves específicas de Redis.  3. **Devolver un valor**: El método devuelve true si la operación fue exitosa, de lo contrario, devuelve false. |
| **Observaciones** | * - En la declaración de métodos no se deberá utilizar caracteres como: * Letra Ñ o ñ. * Caracteres especiales como ¡, ^, #, $, %, &, /, (, ), ¿, ‘, +, -, \*, {, }, [, ], \_. * - Caracteres tildados como á, é, í, ó, ú. * - Redis no tiene una estructura de documentos como MongoDB, por lo que la información se guarda en claves específicas utilizando el sistema clave-valor o hashes. |
| **Ejemplo** | php <br> $objNal->guardarRedis($idUsuario, $nombre, $correo, $clave); <br>  Este método se utiliza para guardar un usuario en Redis. La contraseña es almacenada de manera segura gracias a la encriptación antes de ser guardada. El método devuelve true si la operación fue exitosa y false en caso de error. |

## Declaración de funciones

| **Título** | **Descripción** |
| --- | --- |
| **Sintaxis** | public function cerrarSesion() |
| **Descripción** | El comando para cerrar la sesión en Redis elimina las claves asociadas a las variables de sesión del usuario, como identity y admin. Esto asegura que el estado de la sesión se borre, y cualquier dato relacionado con la autenticación se elimine. Al realizar esto, el usuario queda desconectado, y la aplicación puede redirigirlo 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()

# 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.