

**UNIVERSIDAD PRIVADA DE TACNA FACULTAD DE INGENIERÍA**

**Escuela Profesional de Ingeniería de Sistemas**

***Sistema Municipal de Registro y Transparencia de Establecimientos en Materia de Salubridad - SIRESA***

***Ancco Suaña, Bruno Enrique (2023077472)***  
***Huaman Rivera, Roberto Carlos  (2021071077)***

***Rivera Muñoz, Augusto Joaquin (2022073505)***

**Tacna- Perú 2025**

***Sistema Municipal de Registro y Transparencia de Establecimientos en Materia de Salubridad - SIRESA***

**Estándares de Programación**

**Versión 1.0**

**ÍNDICE GENERAL**

**1. INTRODUCCIÓN**

    1.1. Propósito, Alcance y Objetivos

    1.2. Principios Generales de Programación

**2. ESTÁNDARES DE CODIFICACIÓN JAVA (BACKEND)**

    2.1. Estructura y Nomenclatura de Paquetes

    2.2. Convenciones de Nomenclatura (Clases, Métodos y Variables)

    2.3. Formato y Estructura del Código

    2.4. Comentarios y Documentación Javadoc

    2.5. Arquitectura y Patrones de Diseño (MVC, DAO)

    2.6. Manejo de Excepciones

**3. ESTÁNDARES PARA LA BASE DE DATOS (SQL)**

    3.1. Nomenclatura de Objetos (Tablas, Columnas, Claves)

    3.2. Formato y Estilo para Consultas SQL

**4. ESTÁNDARES PARA EL FRONTEND**

    4.1. Estructura HTML y CSS (Bootstrap)

    4.2. Estándares de JavaScript y Uso de Librerías

**5. ESTÁNDARES DE SEGURIDAD**

    5.1. Manejo de Credenciales y Sesiones

    5.2. Prevención de Inyección SQL y Validación de Datos

**6. CONTROL DE VERSIONES CON GIT**

    6.1. Flujo de Trabajo y Formato de Mensajes de Commit

**7. ANEXOS**

    7.1. Glosario de Términos

    7.2. Herramientas de Desarrollo

**1. INTRODUCCIÓN**

**1.1. Propósito, Alcance y Objetivos**

**Propósito:** Este documento establece las directrices, reglas y buenas prácticas que rigen el desarrollo de software para el proyecto SIRESA. Su finalidad es garantizar la consistencia, calidad, legibilidad y mantenibilidad del código fuente y de la base de datos.

**Alcance:** Los estándares aquí definidos aplican a todas las capas del proyecto SIRESA, incluyendo el código del backend (Java), la base de datos (MySQL), y el frontend (JSP, HTML, CSS, JavaScript).

**Objetivos:**

* Mejorar la calidad general del software.
* Facilitar la colaboración y la incorporación de nuevos miembros al equipo.
* Reducir la complejidad y el tiempo de mantenimiento.
* Asegurar un nivel consistente de seguridad y rendimiento en la aplicación.

**1.2. Principios Generales de Programación**

El desarrollo de SIRESA se adhiere a los siguientes principios de ingeniería de software:

* **KISS (Keep It Simple, Stupid / Mantenlo Simple):** Se prefiere la solución más simple y clara posible. Se evita la complejidad innecesaria.
* **DRY (Don't Repeat Yourself / No te Repitas):** Se evita la duplicación de código. La lógica común se encapsula en métodos o clases reutilizables.

**2. ESTÁNDARES DE CODIFICACIÓN JAVA (BACKEND)**

**2.1. Estructura del Proyecto y Archivos Principales**

Los paquetes se organizan siguiendo una estructura de dominio inverso y deben estar en minúsculas. A continuación, se detallan los paquetes y los archivos clave dentro de ellos.

|  |  |  |
| --- | --- | --- |
| **Paquete** | **Archivos Principales de Ejemplo** | **Propósito del Paquete** |
| com.siresa.config | Conexion.java | Clases de configuración de la aplicación, como la conexión a la base de datos. |
| com.siresa.controlador | ControladorUsuario.java, ControladorEstablecimiento.java | Servlets que actúan como controladores, manejando las solicitudes HTTP y orquestando la lógica de la aplicación. |
| com.siresa.interfaces | CRUDUsuario.java, CRUDEstablecimiento.java | Interfaces que definen los contratos (métodos) para las operaciones de acceso a datos (CRUD). |
| com.siresa.modelo | Usuario.java, Establecimiento.java, Inspeccion.java | Clases de entidad (POJOs/JavaBeans) que representan los objetos de datos del sistema. |
| com.siresa.modeloDAO | UsuarioDAO.java, EstablecimientoDAO.java | Clases de Acceso a Datos (DAO) que implementan las interfaces y contienen la lógica SQL para interactuar con la BD. |
| com.siresa.util | EmailService.java | Clases de utilidad que proporcionan funcionalidades transversales, como el envío de correos electrónicos. |

**2.2. Convenciones de Nomenclatura Detalladas**

**2.2.1. Clases e Interfaces**

* **Clases:** UpperCamelCase (PascalCase). Deben ser sustantivos.
  + *Ejemplo:* Establecimiento, ControladorReporte, EmailService.
* **Interfaces:** UpperCamelCase. Deben describir la capacidad.
  + *Ejemplo:* CRUDUsuario, CRUDInspeccion.

**2.2.2. Métodos y Funciones Principales**

* **Convención:** lowerCamelCase. Deben ser verbos o frases verbales.
* **Categorías Principales:** Los métodos del proyecto se pueden clasificar de la siguiente manera:

|  |  |  |
| --- | --- | --- |
| **Categoría de Método** | **Ejemplos de Métodos** | **Propósito General** |
| **Acciones de Controlador** | doGet(), doPost(), processRequest(), registrarInspeccion(), ejecutarEliminacionLogica() | Orquestar la lógica de una solicitud HTTP. Reciben request y response, interactúan con el Modelo (DAOs) y seleccionan una Vista (JSP) para la respuesta. |
| **Métodos DAO (CRUD)** | obtenerPorId(), listarTodos(), agregar(), actualizar(), softDelete(), contarPorEstado() | Implementar la lógica de acceso a datos. Ejecutan consultas SQL contra la base de datos para crear, leer, actualizar o eliminar registros. |
| **Métodos de Conexión** | conectar(), desconectar(), cerrarRecursos() | Gestionar el ciclo de vida de la conexión a la base de datos y otros recursos JDBC para prevenir fugas. |
| **Métodos de Servicio/Utilidad** | EmailService.enviarCorreo() | Encapsular una funcionalidad específica y reutilizable que no pertenece directamente a la lógica de negocio principal, como enviar un correo. |

**2.2.3. Variables, Parámetros y Constantes**

Se establecen convenciones específicas para cada capa de la aplicación para mantener la claridad.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo de Variable** | **Convención** | **Ejemplo** | **Contexto de Uso** |
| **Variables de Instancia/Locales** | lowerCamelCase | String nombreUsuario; | Declaraciones generales en clases y métodos. |
| **Constantes** | UPPER\_SNAKE\_CASE | static final int MAX\_LOGIN\_ATTEMPTS = 5; | Valores fijos que no cambian. |
| **Instancias de DAOs** | lowerCamelCase | EstablecimientoDAO establecimientoDAO; | En la declaración de campos de los Servlets. |
| **Parámetros de Request** | lowerCamelCase | String motivo = request.getParameter("txtMotivo"); | Dentro de los métodos de un Servlet para capturar datos de formularios HTML. |
| **Atributos de Sesión/Request** | lowerCamelCase | session.setAttribute("usuarioLogueado", usr); | Para pasar datos del Controlador a la Vista (JSP). |
| **Variables en JSPs** | lowerCamelCase | <% Usuario user = (Usuario) session.getAttribute("usuarioLogueado"); %> | Para recuperar y manejar objetos en la capa de presentación. |
| **Objetos JDBC** | Nombres cortos y consistentes | Connection con;, PreparedStatement ps; | En las clases DAO para manejar la conexión y las consultas. |
| **Cadenas SQL** | lowerCamelCase + Sql | String selectAllSql = "SELECT \* FROM ..."; | En las clases DAO para definir las consultas. |

**2.3. Formato y Estructura del Código**

* **Indentación:** 4 espacios.
* **Llaves:** La llave de apertura { se coloca en la misma línea que la declaración. La de cierre } en una nueva línea.
* **Líneas en blanco:** Se utilizan para separar métodos y secciones lógicas dentro de un método para mejorar la legibilidad.

**2.4. Comentarios y Documentación Javadoc**

* Toda clase y método público debe contar con un bloque Javadoc que explique su función, parámetros (@param), valor de retorno (@return) y excepciones (@throws).
* /\*\*
* \* Realiza un borrado lógico de un establecimiento y registra el evento.
* \* @param idEstablecimiento El ID del establecimiento a desactivar.
* \* @param idUsuarioModifico El ID del administrador que realiza la acción.
* \* @param motivo La justificación para la eliminación.
* \* @return true si la operación fue exitosa, false en caso contrario.
* \*/
* public boolean softDelete(int idEstablecimiento, int idUsuarioModifico, String motivo) { ... }

**2.5. Arquitectura y Patrones de Diseño (MVC, DAO)**

La aplicación sigue estrictamente la arquitectura **MVC Modelo 2** y el patrón de diseño **DAO** para garantizar una clara separación de responsabilidades:

* **Modelo:** Encapsula los datos (modelo) y la lógica de acceso a datos (modeloDAO).
* **Vista:** Renderiza la interfaz de usuario (archivos .jsp).
* **Controlador:** Maneja el flujo de la aplicación (Servlets en controlador).

**2.6. Manejo de Excepciones**

* Las excepciones se capturan en bloques try-catch. Como mínimo, la traza de la excepción se registra en la consola del servidor (e.printStackTrace();) para facilitar la depuración.
* Los recursos como Connection, PreparedStatement y ResultSet se gestionan con bloques try-with-resources o se cierran explícitamente en un bloque finally.

**3. ESTÁNDARES PARA LA BASE DE DATOS (SQL)**

**3.1. Nomenclatura de Objetos (Tablas, Columnas, Claves)**

* **Tablas:** UpperCamelCase (ej. Historial\_Cambios\_Estado, Usuarios).
* **Columnas:** snake\_case (minúsculas con guion bajo, ej. id\_usuario, fecha\_registro).
* **Claves Primarias (PK):** Nombradas id\_<tabla\_singular>.
* **Claves Foráneas (FK):** Nombradas id\_<tabla\_referenciada\_singular>.

**3.2. Formato y Estilo para Consultas SQL**

* Las palabras clave de SQL (SELECT, FROM, WHERE, JOIN, etc.) se escriben en mayúsculas.
* Las consultas complejas se dividen en múltiples líneas, indentando las cláusulas JOIN y ON.

**4. ESTÁNDARES PARA EL FRONTEND**

**4.1. Estructura de Archivos JSP y HTML Semántico**

* Se utiliza HTML5 semántico (<nav>, <main>, <section>).
* El siguiente cuadro detalla los archivos JSP clave y su propósito:

|  |  |
| --- | --- |
| **Archivo JSP** | **Propósito Principal** |
| login.jsp | Muestra el formulario de inicio de sesión y maneja mensajes de error/logout. |
| admin\_dashboard.jsp | Panel de control principal para el rol de Administrador. |
| inspector\_dashboard.jsp | Panel de control principal para el rol de Inspector. |
| ciudadano\_dashboard.jsp | Panel de control principal para el rol de Ciudadano. |
| listarEstablecimientos.jsp | Muestra la lista de establecimientos con filtros y acciones. |
| formEstablecimiento.jsp | Formulario dinámico para registrar o editar un establecimiento. |
| detalleEstablecimiento.jsp | Muestra la información completa y las acciones disponibles para un solo establecimiento. |
| formInspeccion.jsp | Formulario para registrar una nueva inspección a un establecimiento. |
| reportePorEstado.jsp | Muestra el reporte de conteo de establecimientos por estado, incluyendo gráficos. |
| historialGlobalCambios.jsp | Muestra la bitácora de los cambios de estado más recientes en todo el sistema. |

**4.2. Estilos CSS y Uso del Framework Bootstrap**

* El diseño se basa en el framework **Bootstrap 4**.
* Las clases CSS personalizadas usan kebab-case (ej. .report-container).

**4.3. Estándares de JavaScript y Funciones Clave**

* Se utiliza camelCase para variables y funciones. Se prefiere const y let.
* **Funciones Clave:**

|  |  |  |
| --- | --- | --- |
| **Función JavaScript** | **Archivo(s) donde se usa** | **Propósito** |
| initMap(), initMapDetail() | mapaEstablecimientos.jsp, detalleEstablecimiento.jsp | Inicializa los mapas de Google Maps, establece el centro, zoom y marcadores. |
| initMapPicker() | formEstablecimiento.jsp | Inicializa el mapa interactivo para seleccionar coordenadas. |
| updateMarkerAndInputs() | formEstablecimiento.jsp | Actualiza los campos de texto de latitud/longitud y el marcador en el mapa. |
| geocodeLatLng() | formEstablecimiento.jsp | Utiliza la Geocoding API de Google para obtener una dirección a partir de coordenadas. |
| Chart(...) | reportePor\*.jsp | Constructor de la librería Chart.js para crear y renderizar los gráficos de reportes. |

**5. ESTÁNDARES DE SEGURIDAD**

**5.1. Manejo de Credenciales y Sesiones**

* Las credenciales sensibles (BD, correo, APIs) se gestionan a través de variables de entorno del servidor.
* La autenticación se verifica mediante HttpSession en cada página protegida.

**5.2. Prevención de Inyección SQL y Validación de Datos**

* Se utiliza exclusivamente PreparedStatement con parámetros (?) para todas las consultas SQL.
* Los datos de entrada de formularios se validan en el backend antes de ser procesados.

**6. CONTROL DE VERSIONES CON GIT**

**6.1. Flujo de Trabajo y Formato de Mensajes de Commit**

* **Flujo:** Se utiliza un flujo de feature branches. Las nuevas funcionalidades se desarrollan en ramas separadas y se fusionan a main tras su aprobación.
* **Mensajes:** Los mensajes de commit son descriptivos y usan prefijos como feat:, fix:, docs:.

**7. ANEXOS**

**7.1. Glosario de Términos**

* **DAO:** Data Access Object.
* **MVC:** Modelo-Vista-Controlador.
* **WAR:** Web Application Archive.

**7.2. Herramientas de Desarrollo**

* **IDE:** Apache NetBeans
* **Lenguaje:** Java (JDK 17)
* **Servidor de Aplicaciones:** Apache Tomcat / Google App Engine
* **Base de Datos:** MySQL Server / Google Cloud SQL
* **Control de Versiones:** Git
* **Plataforma de Despliegue:** Google Cloud Platform (App Engine)