ESTÁNDARES DE DESARROLLO DEL SISTEMA DE INGRESO DE CHOFERES DE LA EPN

Versión 1.0

Escuela Politécnica Nacional

Facultad de Ingeniería en Sistemas

2024

CONTENIDO:

1. Propósito

El propósito de este documento es establecer un conjunto de estándares de codificación para el desarrollo del sistema de gestión de información de choferes de buses de la Escuela Politécnica Nacional. Estos estándares buscan garantizar la coherencia, la calidad y la mantenibilidad del código a lo largo de todo el ciclo de vida del proyecto. Además, este documento servirá como una guía para todos los desarrolladores involucrados, asegurando que el código cumpla con los mejores estándares de la industria y facilite la colaboración y la revisión del código.

2. Ámbito

- Base de Datos: Reglas para la estructuración y nombrado de tablas, columnas y relaciones, así como prácticas recomendadas para la optimización de consultas y la integridad de los datos.
- Introducción de Datos: Procedimientos y estándares para la captura y almacenamiento de datos dentro del sistema, asegurando que se mantenga la integridad y la seguridad de la información.
- Variables y Clases dentro del Código General del Programa: Convenciones de nomenclatura y estructuración de código para variables, funciones, métodos, clases y otros elementos de programación, para asegurar la claridad y consistencia en la implementación del software.

3. Convenciones de Nomenclatura

Las convenciones de nomenclatura son esenciales para mantener la claridad y consistencia del código. Estas convenciones ayudarán a los desarrolladores que se involucren en este proyecto a entender y colaborar eficazmente en el proyecto sin dañar los estándares y la nomenclatura con la que se escribió en un inicio.

3.1 Variables

Utilizar camelCase para nombrar las variables. Los nombres deben ser descriptivos y representar claramente el propósito de la variable.
Ejemplo: nombreConductor, marcaCarro

3.2 Funciones y Métodos

Igualmente, el estándar camelCase también se utiliza para funciones y métodos. Los nombres deben describir claramente la acción que realiza la función o método.
Ejemplo: calcularEdad(), obtenerInformacionConductor()

3.3 Clases y Objetos

Usar PascalCase (también conocido como UpperCamelCase) para nombres de clases y objetos. Esto implica iniciar cada palabra con una letra mayúscula.

3.4 Archivos y Directorios

Los nombres de archivos y directorios deben ser claros y descriptivos, y deben reflejar su contenido o propósito en el proyecto. Se recomienda mantener consistencia en el uso de minúsculas y separar las palabras mediante el uso de guiones medios o puntos.
Ejemplo: conductor_datos.js, rutas_datos.py

Estas convenciones deben ser seguidas estrictamente para facilitar la lectura, mantenimiento y escalabilidad del código. Además, estas prácticas ayudan a evitar errores y confusiones que pueden surgir debido a inconsistencias en la nomenclatura.

4. Estructura del código

4.1 Organización del Proyecto

Arquitectura del Proyecto: El proyecto sigue una arquitectura en capas que separa claramente las responsabilidades de negocio, acceso a datos, y la interfaz de usuario (UI). Esto se refleja en las siguientes capas:

- ➔ `businessLogical`: Contiene la lógica de negocio, encapsulada en clases como `GestorChofer` y `GestorModificaciones`.
- ➔ `dataAccesComponent`: Encargada de la interacción con la base de datos. Aquí residen los DAO (`Data Access Object`), responsables de las operaciones CRUD.
- ➔ `ui`: Maneja la presentación y la interacción con el usuario, dividida en formularios y componentes gráficos.

Componentes y Jerarquía

- ➔ Los DAOs implementan una interfaz común (`IDAO`) que define las operaciones básicas de acceso a datos, promoviendo la reutilización y mantenimiento del código.
- ➔ Las entidades (`entity`) representan tablas de la base de datos y están diseñadas siguiendo el patrón de objeto relacional (ORM). Cada entidad refleja fielmente la estructura de sus tablas correspondientes.
- ➔ Los formularios (`forms`) y componentes gráficos están diseñados para seguir el patrón MVC (Modelo-Vista-Controlador), asegurando una separación clara entre la lógica de la UI y la lógica de negocio.

4.2 Convenciones de Nomenclatura

Nombres Descriptivos:Se utilizan nombres descriptivos para las clases y métodos, siguiendo las convenciones de `PascalCase` para clases y `camelCase` para métodos y variables. Los nombres de las clases en la capa de negocio (`businessLogical`) y acceso a datos (`dataAccesComponent`) están directamente relacionados con su funcionalidad y propósito dentro del sistema.

Ejemplo: `GestorChofer`, `ChoferDAO`, `RegistroDatosForm`.

Archivos y Directorios:

- ➔ Los directorios están organizados según su funcionalidad dentro del sistema, utilizando nombres claros y descriptivos.
- ➔ Los nombres de los archivos reflejan su propósito específico dentro del sistema y mantienen consistencia con las convenciones de codificación.

4.3 Estructura y Estilo de Código

Indentación y Espaciado: Todo el código sigue una convención de 4 espacios por nivel de indentación. Esto asegura que la estructura del código sea clara y fácilmente legible.

Ejemplo:

```
public class GestorChofer {
    public void registrarNuevoChofer(Chofer chofer) throws SQLException {
        if (chofer.esValido()) {
        choferDAO.insert(chofer);
        } else {
        throw new ValidationException("Datos del chofer no válidos.");
        }
    }
}
```

Longitud de Líneas: Se ha establecido un límite de 100 caracteres por línea para evitar el desbordamiento horizontal y mantener la legibilidad. Cuando es necesario, se utiliza la continuación de línea.

Ejemplo:

```
public boolean actualizarChofer(int idChofer, String cedula, String nombres,
                String apellidos, String telefono, String huella) {
}
```

Comentarios y Documentación: Se emplean comentarios Javadoc para documentar las clases, métodos y atributos. Los comentarios en línea se utilizan para explicar secciones de código complejas o poco intuitivas.

Ejemplo:

```
/**
* Registra un nuevo chofer en la base de datos.
*
* @param chofer Objeto Chofer con los datos del nuevo chofer.
* @throws SQLException si ocurre un error al registrar el chofer.
*/
public void registrarNuevoChofer(Chofer chofer) throws SQLException {
        // Validar datos antes de registrar
        if (chofer.esValido()) {
                choferDAO.insert(chofer);
        } else {
                throw new ValidationException("Datos del chofer no válidos.");
        }
}
```

5. Prácticas de codificación

5.1 Control de Versiones

Gestión con Git: Se utiliza Git para el control de versiones del proyecto. Cada funcionalidad o corrección de errores se desarrolla en una rama separada con nombres descriptivos (por ejemplo, `feature/nuevo-formulario` o `bugfix/correccion-dao`). Los commits deben ser concisos y descriptivos, utilizando la estructura `[Tipo]: Descripción breve del cambio.

Ejemplo de mensajes de commit:

git ci “añadir validación de datos en GestorChofer”

git ci “corregir error en la inserción de registros de chofer”

5.2 Gestión de Errores y Excepciones

Manejo de Excepciones: Se implementan bloques `try-catch` para manejar las excepciones, especialmente en operaciones críticas como el acceso a la base de datos. Los mensajes de error proporcionan información clara al usuario y registran el error para un análisis posterior.

Ejemplo:

```
try {
    gestorChofer.registrarNuevoChofer(chofer);
} catch (SQLException e) {
    log.error("Error al registrar el nuevo chofer: " + e.getMessage());
    mostrarMensajeError("No se pudo registrar el chofer, por favor intente nuevamente.");
}
```

5.3 Pruebas

Pruebas Unitarias y de Integración: Se desarrollan pruebas unitarias para asegurar que cada método funcione correctamente, utilizando frameworks como JUnit. Las pruebas de integración se aplican para verificar que diferentes módulos del sistema funcionen de manera coherente cuando se integran.

Ejemplo de Prueba Unitaria:

```
@Test
public void testRegistrarNuevoChofer() {
    Chofer chofer = new Chofer("1234567890", "Juan", "Perez", ...);
    boolean registrado = gestorChofer.registrarNuevoChofer(chofer);
    assertTrue(registrado);
}
```

5.4 Documentación

Documentación Técnica: Se recomienda el uso de herramientas como JSDoc, Swagger, o similar para documentar APIs y funciones importantes. Además, se mantiene un archivo `README.md` actualizado que incluye instrucciones de instalación, configuración y ejecución del proyecto.

6. Estándares de seguridad

La seguridad es un aspecto crítico en el desarrollo de software, especialmente cuando se manejan datos sensibles como la información personal de los choferes de buses o de los administradores de la base de datos. A continuación se presentan los estándares de seguridad que se deben seguir para proteger los datos y garantizar la integridad del sistema.

6.1 Validación y Sanitización de Datos

Validación de Entradas: Todos los datos de entrada, especialmente aquellos provenientes de usuarios o sistemas externos, deben ser validados rigurosamente para asegurar que cumplan con los formatos esperados y límites establecidos. Todo esto se realiza dentro del programa mismo, pues dentro de clases clave (como por ejemplo, los formularios) se debe realizar siempre validaciones para que el usuario no tenga ni siquiera la oportunidad datos que no sean admitidos dentro de los estándares. Por ejemplo, en la sección de los formularios existen varios validadores para que en las variables de la cédula o número de teléfono no pueda ingresar caracteres que no sean numéricos.

Sanitización de Datos: Antes de almacenar o procesar los datos, se debe eliminar cualquier contenido potencialmente malicioso. Esto incluye la eliminación de caracteres especiales que podrían ser utilizados para inyecciones SQL u otras vulnerabilidades.

6.2 Autenticación y Autorización

Autenticación Segura: En el caso de los administradores, se recomienda implementar métodos de autenticación robustos, como el uso de contraseñas fuertes o nombres de usuario que no sean fáciles de descifrar.

Autorización: Para los desarrolladores y programadores que van a manipular el código y crear su propio usuario de administrador, se recomienda que solo los usuarios autorizados tengan acceso a ciertas funciones y datos del sistema. Esto implica no dar credenciales a usuarios que no tengan que ver con el proceso de manipulación y mantenimiento del código.

6.3 Manejo de Errores y Excepciones

Manejo Seguro de Errores: Capturar y manejar adecuadamente todos los errores y excepciones, sin exponer detalles internos del sistema en los mensajes de error. Los mensajes de error deben ser genéricos y no revelar información sensible.
Ejemplo: En lugar de mostrar detalles de un error de base de datos, mostrar un mensaje como "Ocurrió un error. Por favor, inténtelo de nuevo más tarde."

Estos estándares de seguridad son esenciales para proteger la integridad y confidencialidad de los datos, así como para asegurar la confianza de los usuarios en el sistema.

7. Estándares de Bases de Datos

7.1 Nombres de Tablas y Columnas

Convención de Nombres: Los nombres de tablas y columnas siguen la convención `snake_case` y están en singular, lo que facilita la identificación y la comprensión de su propósito. Los nombres deben ser descriptivos y evitar abreviaturas innecesarias.

Ejemplo en SQL:

```
CREATE TABLE `vehiculo` (
    `vehiculo_id` INT(10) NOT NULL AUTO_INCREMENT,
    `chofer_id` INT(10) NOT NULL,
    `vehiculo_placa` VARCHAR(8) NOT NULL,
    `vehiculo_tipo` VARCHAR(50) NOT NULL,
    `vehiculo_marca` VARCHAR(50) NOT NULL,
```

```
`vehiculo_modelo` VARCHAR(50) NOT NULL,
`estado` CHAR(1) NOT NULL DEFAULT 'A',
    PRIMARY KEY (`vehiculo_id`),
    UNIQUE INDEX `unq_vehiculo_placa` (`vehiculo_placa`)
    );
```

7.2 Integridad Referencial

Claves Primarias y Foráneas: Todas las tablas deben tener una clave primaria (`PRIMARY KEY`) que asegure la unicidad de los registros. Además, se deben utilizar claves foráneas (`FOREIGN KEY`) para establecer y mantener la integridad referencial entre tablas relacionadas.

Ejemplo en SQL:

```
    ALTER TABLE `vehiculo`
    ADD CONSTRAINT `fk_vehiculo_chofer`
FOREIGN KEY (`chofer_id`) REFERENCES `chofer`(`chofer_id`)
    ON DELETE NO ACTION ON UPDATE NO ACTION;
```

Restricciones y Validaciones: Se implementan restricciones como `CHECK` para garantizar que los datos ingresados en la base de datos cumplan con las condiciones establecidas.

Ejemplo en SQL:

```
    CONSTRAINT `chk_chofer_estado`
    CHECK ((`estado` in ('A', 'X')))
```

7.3 Optimización y Rendimiento

Índices: Se recomienda el uso de índices en las columnas que se utilizan frecuentemente en condiciones `WHERE` o en las relaciones de claves foráneas. Esto optimiza el rendimiento de las consultas, especialmente en bases de datos grandes.

Ejemplo en SQL:

```
CREATE INDEX `idx_chofer_cedula`

ON `chofer` (`chofer_cedula`);
```

Consultas SQL: Las consultas SQL deben estar optimizadas para evitar subconsultas innecesarias y asegurar que las tablas estén correctamente normalizadas. Además, se considera el uso de procedimientos almacenados para operaciones complejas que se repiten frecuentemente.

7.4 Normalización

Primera, Segunda y Tercera Forma Normal (1NF, 2NF, 3NF): La estructura de la base de datos debe estar normalizada hasta al menos la tercera forma normal para eliminar redundancias y garantizar la integridad de los datos.

- ➔ 1NF: Todos los atributos deben contener valores atómicos.
- ➔ 2NF: Todos los atributos no clave deben depender completamente de la clave primaria.
- ➔ 3NF: Todos los atributos no clave deben depender únicamente de la clave primaria, no de otros atributos.

8. Cambios, Revisiones y Aprobaciones

Control de Cambios

Fecha	**Autor**	**Versión**	**Referencia del cambio**
2024-08-06	ruben.cuenca	1.0	Creación del documento

Revisiones

Fecha	**Autor**	**Versión**	**Referencia del cambio**

Aprobaciones

Fecha	**Autor**	**Versión**	**Referencia del cambio**

9. Anexos

Para el desarrollo de este proyecto, se utilizaron las siguientes herramientas y tecnologías:

9.1 Lenguaje de Programación:

Java: Utilizado como el lenguaje principal para la lógica del servidor y la implementación del sistema.

9.2 Entorno de Desarrollo:

Visual Studio Code (VS Code): El entorno de desarrollo integrado (IDE) elegido para escribir, depurar y gestionar el código del proyecto. Se han utilizado diversas extensiones para mejorar la productividad y el manejo de la base de datos.

9.3 Manejo de Base de Datos:

Laragon: Una herramienta de desarrollo local que proporciona un entorno fácil de configurar para servidores web, bases de datos y otras utilidades. Se utiliza para el manejo de la base de datos en el entorno de desarrollo.

10. Aprobación del documento

Los firmantes certifican la información presentada en este documento

Elaborado por:

_____

Rubén Cuenca
Santiago De La Cruz
Estudiantes de la Escuela Politécnica Nacional

Revisado por:

_____

Patricio Paccha
Especialista de desarrollo