UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERÍA

Escuela Profesional de Ingeniería de Sistemas

Proyecto Sistema web de plataforma de voluntariado "NeedU"

Curso: *Programación III*

Docente: *Elard Rodriguez Marca*

Integrantes:

Cutipa Gutierrez, Ricardo Daniel (2021069827)

Apaza Calle, Albert Kenyi (2021071075)

Churacutipa Blas, Erick Scott (2020067578)

Tacna – Perú

2023

Sistema *web de Plataforma de Voluntariado“NeedU”*

Documento de Arquitectura de Software

Versión *{1.2}*

HISTORIAL DE REVISIÓN

Historia de Revisión				
Item	Fecha	Versión	Descripción	Equipo
1	04/12/23	1.0	Version 1.0	G02

Estandares de Programacion

1. OBJETIVO

El objetivo de este documento es establecer un conjunto coherente y eficiente de estándares y prácticas de programación para el desarrollo y mantenimiento del sistema de gestión de voluntariado "NeedU". Estos estándares están diseñados para garantizar la calidad, la seguridad, y la eficiencia del código, así como facilitar la colaboración entre diferentes desarrolladores y equipos de trabajo.

Este documento abarca convenciones sobre nomenclatura, estructura de la base de datos, estilo de codificación, manejo de errores, documentación del código, y procedimientos para el control de versiones. Al adherirse a estos estándares, el equipo de "NeedU" podrá desarrollar un software robusto, escalable, y fácil de mantener, lo cual es crucial para el éxito a largo plazo del proyecto.

ALCANCE

El presente documento establece estándares y prácticas que son aplicables a todas las etapas del ciclo de vida del desarrollo de software del proyecto "NeedU". Este alcance detallado asegura que todos los aspectos del desarrollo estén cubiertos de manera integral:

Diseño y Modelado de la Base de Datos:

- Establece las normas para el diseño estructural de la base de datos, incluyendo la definición de tablas, relaciones, y esquemas de normalización.
- Orientaciones sobre la implementación de medidas de seguridad y rendimiento para el manejo de datos.

Desarrollo del Backend y Frontend:

- Directrices para la arquitectura del backend, incluyendo la lógica de negocio, la gestión de APIs y la integración con la base de datos.
- Estándares para el desarrollo del frontend, enfocándose en la usabilidad, accesibilidad y diseño responsivo.

Pruebas de Software:

- Protocolos para pruebas unitarias y de integración, asegurando que cada componente funcione correctamente tanto de manera independiente como en conjunto.
- Estrategias para pruebas funcionales y no funcionales, incluyendo pruebas de rendimiento, seguridad y compatibilidad.

Mantenimiento y Actualizaciones del Sistema:

- Procedimientos para la actualización regular y el mantenimiento del sistema, garantizando su relevancia y eficiencia a lo largo del tiempo.

- Directrices para la documentación y el seguimiento de cambios, facilitando futuras actualizaciones y depuraciones.

2. Declaración de Variables

Para asegurar la claridad y eficiencia en el manejo de variables en el proyecto "NeedU", se establecen las siguientes normas para su declaración:

Longitud y Descripción

- **Longitud Máxima:**
    - Las variables deben tener una longitud máxima de 16 caracteres para equilibrar claridad y manejabilidad.
- **Descripción:**
    - El nombre debe ser suficientemente descriptivo para identificar claramente su propósito.

Alcance	Prefijo	Ejemplo
Controlador	Ctrl	CtrlOportunidad
Modelo	Mdl	MdlOportunidad
DAO	Dao	DaoOportunidad
Servlet	Srv	SrvOportunidad
Sesión	Ses	SesUsuario
Petición	Req	ReqParametro
Respuesta	Res	ResResultado

Tipo de Dato de la Variable

El tipo de dato es un componente crucial para la identificación rápida de las características de la variable:

Tipo de Dato	Prefijo	Ejemplo
String	str	StrNombreUsuario
Integer	int	IntIdUsuario
Boolean	bool	BoolEsValido
Objeto	obj	ObjConexion

Estructura de la Variable

La estructura de la variable sigue la convención de CamelCase:

FORMATO: La primera palabra en minúscula seguida de la segunda palabra con la inicial en mayúscula.

EJEMPLO: strNombreUsuario, intIdUsuario, boolEsValido, objConexion.

Ejemplos de Declaración de Variables

ControladorOportunidad.java:

```
OrganizacionDAO ctrlOrgDao = new OrganizacionDAO();
HttpServletRequest reqRequest;
HttpServletResponse resResponse;
```

OportunidadDAO.java:

Conexion daoConexion = new Conexion();

PreparedStatement daoPs;

ResultSet daoRs;

Siguiendo estas directrices, el código dentro del proyecto "NeedU" será coherente, legible y fácil de mantener, asegurando que las variables estén claramente definidas y documentadas.

2.1 Descripción de la Variable

El nombre asignado a cada variable será representativo de su propósito y estará estrechamente relacionado con la funcionalidad para la que se crea.

Ejemplos:

idCuenta: Identificador único de la cuenta de usuario.

tipoEstado: Representa el estado de un objeto o entidad.

instalacion: Referencia al proceso o estado de instalación.

VARIABLES LOCALES:

Librerías:

- mysql-connector-j-8.0.33.jar
- itextpdf-5.5.9.jar
- gson-2.10.1.jar

Login:

txtCorreo: Campo de texto para ingresar el correo electrónico del usuario.

txtContraseña: Campo de texto para ingresar la contraseña del usuario.

txtCaptcha: Campo de texto para ingresar la validación del captcha.

btnIniciarSesion: Botón utilizado para iniciar la sesión.

Textos:

JtextField: Componente de la interfaz gráfica para la entrada de texto.

2.2 Variables de Tipo Arreglo

En el contexto del proyecto "NeedU", el manejo de colecciones de datos se realizará mediante la utilización de variables tipo arreglo o lista, según corresponda. Estas variables permiten almacenar múltiples valores o instancias de objetos bajo un mismo identificador de variable. La nomenclatura a seguir será la siguiente:

Formato:

El nombre de la variable debe comenzar con el prefijo lista, seguido del tipo de objeto que contiene, todo en notación CamelCase.

Ejemplos:

List<Map<String, Object>> listaOportunidades:

Esta variable representa una lista de mapas, donde cada mapa contiene pares clave-valor que representan las propiedades de una oportunidad.

3. **Declaración de Variables**

En el desarrollo de interfaces de usuario, es crucial nombrar los controles de manera que su tipo y propósito sean inmediatamente reconocibles. A continuación, se detalla la convención para la asignación de nombres a los controles de la aplicación "NeedU".

3.1 Tipo de datos

Para cada control, se asignará un tipo de dato correspondiente al propósito de dicho control. Los tipos de datos comunes y sus mnemónicos son:

Tipo de variable	Mnemónico	Descripción
Byte		Entero de 8 bits sin signo.
Integer		Entero de 32 bits con signo.
Char		Un carácter UNICODE de 16 bits.
String		Cadena de caracteres.
Date		Formato de fecha/hora.
Timestamp		Representar fecha y hora.
Boolean		Valor lógico: verdadero y falso.
Float		Coma flotantes, 11-12 dígitos significativos.
Double		Coma flotante, 64 bits (15-

		16 dígitos significativos).
Object		Objeto genérico.

3.2 Prefijo para el Control

El prefijo del control se establece mediante una combinación de letras, preferiblemente consonantes, que identifican de forma única el tipo de control. Por ejemplo, para un botón (Button), se utilizará el prefijo btn.

3.3 Nombre descriptivo del Control

El nombre de cada control se complementará con una descripción clara y concisa de su función en la aplicación. La descripción debe ser específica para evitar ambigüedades.

Tipo de Control	Prefijo	Ejemplo
JLabel	lbl	lblUsuario
JTextField	txt	txtUsuario
JTextField	txt	txtClave
JButton	btn	btnIniciar

3.4 Nombre descriptivo del Control

Título	Descripción
Sintaxis	Se debe declarar una sola variable por línea para mejorar la legibilidad del código. Formato: [TipoVariable] [NombreVariable]
Longitud	**Todas las variables o atributos tendrán una longitud máxima de 30 caracteres.**
Nomenclatura	**- El nombre de la variable puede incluir más de un sustantivo; en tal caso, se escribirán juntos, utilizando CamelCase para separar palabras. - Si se tuvieran variables que puedan tomar nombres iguales dentro de un mismo ámbito, se les agregará un número al final para diferenciarlas. Este número será correlativo si las variables están dentro de un mismo método.**
Observaciones	**No se deben utilizar caracteres especiales o acentuados en la declaración de variables o atributos. Esto incluye pero no se limita a: Ñ/ñ, caracteres especiales como ¡, ^, #, $, %, &, /, (,), ¿, ', +, -, *, {, }, [,], y vocales tildadas (á, é, í, ó, ú).**

3.5 Nombre descriptivo del Control

Título	Descripción
Sintaxis	`Cls [Nombre de Clase]`
Descripción	**El nombre de las clases tendrá una longitud máxima de 30 caracteres y las primeras letras de todas las palabras estarán en mayúsculas. Tipo se refiere a si la clase será: Private, Public o Protected.**

Observaciones	**No usar caracteres especiales o acentuados: Ñ/ñ, ¡, ^, #, $, %, &, /, (,), ¿, ', +, -, *, {, }, [,], y vocales tildadas (á, é, í, ó, ú).**
Ejemplo	`public class Oportunidad` **Indica una clase Oportunidad**

3.6 Declaración de Métodos

Título	Descripción
Sintaxis	`Accion+ElementoAfectado[(ListaParámetros)]`
Descripción	El nombre del método consta de hasta 25 caracteres. La primera letra de la primera palabra en minúscula y las siguientes palabras empezarán con letra mayúscula.
Observaciones	No usar caracteres especiales o acentuados: Ñ/ñ, ¡, ^, #, $, %, &, /, (,), ¿, ', +, -, *, {, }, [,], _.
Ejemplo	`public boolean crearOportunidad(Oportunidad opo)` Indica un método para crear una oportunidad

3.7 Control de Versiones de Codigo Fuente

Cada modificación realizada será guardada de la forma:

Título	Descripción
Formato	`[NOMBRE DOCUMENTO]_[FECHA]_[HORA]` (Fecha en formato yyyymmdd y hora en formato HHMM)
Descripción	Se generarán archivos con extensiones .zip o .rar. Ejemplo: `VMRW_25071021_1524.zip`

4. **Clases**

Autodescriptividad: Las clases deben tener nombres autodescriptivos para facilitar la comprensión de su función sin necesidad de revisar el código interno.

Estándar de Nombres: Los nombres de las clases deben iniciar con una letra mayúscula y seguir con CamelCase, reflejando su propósito o la entidad que representan.

Restricciones: No se utilizarán espacios en blanco ni caracteres especiales en los nombres de las clases.

Nombre de Clase	Descripción
Oportunidad	Almacena una instancia de los atributos de la tabla relacionada con las oportunidades.
OrganizacionDAO	Gestiona las operaciones de acceso a datos para las organizaciones.
ControladorOportunidad	Controla las acciones de la aplicación relacionadas con las oportunidades.

5. **Clases**

Los nombres de funciones y procedimientos deben ser autodescriptivos para comprender su función sin necesidad de revisar el código.

El estándar para los nombres de procedimientos es usar un verbo que describa la acción seguido por un sustantivo (objeto sobre el cual actúa).

Se recomienda usar un nombre que represente una acción y un objeto, indicando qué hace el procedimiento "a..." o "con...".

El verbo debe estar en infinitivo.

Mantener consistencia en el orden y uso de las palabras.

No se utilizan espacios en blanco ni caracteres especiales en los nombres.

La nomenclatura de argumentos y valores de retorno sigue las mismas convenciones que las variables.

Métodos y Funciones	**Descripción**
crearOportunidad	Procesa los datos de una oportunidad y devuelve true si se crea con éxito.
listarOportunidades	Lista todas las oportunidades disponibles.
listarOportunidadesPorOrganizacion	Lista oportunidades por organización.

6. **Clases**

- Mejora la Legibilidad del Programa: La documentación adecuada hace que tu programa sea más legible y comprensible. Los comentarios y explicaciones ayudan a otros programadores (o incluso a ti mismo en el futuro) a entender el propósito y el funcionamiento del código.

- Facilita las Modificaciones: Cuando documentas tu programa desde el principio, evitas la necesidad de realizar un profundo estudio cada vez que se necesita hacer una modificación. Esto es especialmente útil ya que no siempre quien modifica el código es la misma persona que lo escribió inicialmente.

- Fomenta la Reutilización: La documentación adecuada permite que los módulos y rutinas de tu programa sean fácilmente reutilizables en otros programas o dentro del mismo proyecto. Los demás desarrolladores pueden comprender cómo usar y adaptar tus componentes.

- Detecta Necesidad de Reescritura: Si encuentras dificultades para explicar el código mediante comentarios, es una señal de que el código podría estar mal escrito. La documentación puede ayudarte a identificar áreas que requieren una revisión y reescritura para mejorar la claridad.

7. Conclusiones

- **Importancia de un Estándar de Programación**: Una programación de calidad y una implementación legible se logran mediante la adopción de un buen estándar o patrón de programación. Estos estándares establecen reglas y convenciones que hacen que el código sea coherente y fácilmente comprensible para todos los miembros del equipo.

- **Conocimiento del Estándar:** Es crucial que los programadores tengan un conocimiento previo del estándar de programación o, en su defecto, que se familiaricen con el documento que lo describe. Esto ayuda a prevenir diferencias en la interpretación de las convenciones.

- **Beneficios de la Documentación**: La documentación no solo crea un código legible, sino que también proporciona una sólida base para futuros desarrollos y mantenimiento del código. Un programa bien documentado es más sostenible a lo largo del tiempo y permite a los equipos trabajar de manera más eficiente.