CONTROL DE CAMBIOS

FECHA	VERSIÓN	DESCRIPCIÓN
18-07-2023	1.0	Creación del Documento Manual de Arquitectura de Desarrollo Seguro para la Secretaría Distrital de Movilidad

AUTORIZACIONES

ELABORÓ	REVISÓ		APROBÓ
	Del Proceso	**Del grupo OAPI**	
Nombre: Adriana Gutiérrez Eliana Robayo	**Nombre:** Roger Alfonso González	**Nombre:** Lady Carolina Cárdenas Pérez	**Nombre:** Yohana Pineda Afanador
Firma: 	**Firma:** 	**Firma:** 	**Firma:** 
Cargo: Profesional Especializado Oficina de Tecnologías de la Información y las comunicaciones. Profesional Contratista Especializado Oficina de Tecnologías de la Información y las comunicaciones	**Cargo:** Profesional Contratista Oficina de Tecnologías de la Información y las comunicaciones	**Cargo:** Profesional Contratista Oficina Asesora de Planeación Institucional	**Cargo:** Jefe de la Oficina de Tecnologías de la Información y las comunicaciones

TABLA DE CONTENIDO

1. INTRODUCCIÓN

Los sistemas de información[1] y aplicaciones de la Secretaría Distrital de Movilidad (SDM) apoyan la operatividad de los procesos y gestionan la información como insumo valioso.

La Entidad cuenta con sistemas de información que son desarrollados internamente, otros desarrollados por terceros o adquiridos a empresas desarrollo de software, por lo cual es necesario estandarizar y establecer los procesos y lineamientos de ingeniería de software.

Actualmente no existe una documentación o disposiciones explícitas para las actividades enmarcadas dentro de un proceso de desarrollo de software, lo que dificulta la unificación y estandarización de los artefactos y producto del desarrollo. Es necesario identificar tipos de actores, participantes, roles, documentación, responsabilidades y modelos de operación para los proyectos de desarrollo de software y el ciclo de vida de las aplicaciones.

1.1. PROPÓSITO

Presentar de una forma explícita y consistente la gobernabilidad sobre el diseño y la implementación de los sistemas de información y aplicaciones de la organización Secretaría Distrital de Movilidad – SDM.

Este documento tiene como propósito, el establecimiento de lineamientos técnicos y estándares para el desarrollo interno o externo de sistemas de información para la SDM. Comprende la información necesaria que involucra el análisis, diseño, desarrollo e implementación de los sistemas de información bajo conceptos y tecnologías recientes.

1.2. ALCANCE

Este documento contiene las directrices y buenas prácticas para asegurar y gestionar el desarrollo y/o implementación la correcta operación y calidad de todos los sistemas de información y aplicaciones de la organización.

2. POLÍTICA DE DESARROLLO DE SOFTWARE

La política de desarrollo de software de la SDM contiene declaraciones, responsabilidades, lineamientos técnicos de arquitectura y desarrollo de software a los cuales se les debe dar cumplimiento.

La Secretaría Distrital de Movilidad a través de la Oficina de Tecnologías de la Información y Comunicaciones – OTIC, se compromete a verificar el cumplimiento y atributos de calidad de software como la modificabilidad, la escalabilidad, la seguridad de

[1] *Conjunto de agentes, códigos y procesos que interactúan coordinadamente entre sí.*

la información, entre otros contemplados en esté manual y en el Manual de Políticas Específicas de Seguridad de la Información PA04-M01, al implantar la presente política de desarrollo de software.

2.1 DECLARACIÓN DE LA POLÍTICA

- La verificación de la aplicación de la política de desarrollo de software está a cargo del equipo de Fábrica de Software perteneciente a la OTIC.

- Las personas involucradas y todos aquellos que tengan responsabilidad sobre el desarrollo de sistemas de información, deben optar los lineamientos contenidos en la presente política con la finalidad de cumplir con los pilares de la seguridad de la información del desarrollo de aplicaciones.

- La OTIC es la responsable de divulgar a las personas involucradas y vinculadas a la SDM para que cumplan con la política de desarrollo de software.

- El incumplimiento total o parcial de la presente política por parte del funcionariado, podrá constituirse como falta disciplinaria y por lo tanto podrá dar lugar a la acción e imposición de la sanción correspondiente establecida en la ley penal, en el código disciplinario único y en las demás normas que se encuentren vigentes.

- El incumplimiento total o parcial de la presente política por parte de las y los contratistas, personal que labore en las instalaciones vinculado con proveedores de la SDM o personal externo (empresa o entidad externa), estará sujeto a las sanciones contractuales, civiles y/o penales a que haya lugar, por los daños y perjuicios generados a la Secretaría Distrital de Movilidad o a terceros.

2.2 RESPONSABILIDADES

- La política de desarrollo de software es de aplicación obligatoria para todo el personal de desarrollo de software que labore en la SDM.

- La SDM y OTIC aprueban esta política y son responsables de la autorización de sus cambios y/o nuevas versiones.

- La persona oficial de seguridad de la información de la OTIC es responsable de impulsar la ejecución y cumplimiento del atributo de calidad: Seguridad de los sistemas de información, incluido en esta política de desarrollo de software.

- El equipo de fábrica de software de la OTIC es el responsable de impulsar la ejecución y cumplimiento de la política de desarrollo de software.

- Las personas responsables del desarrollo, adquisición y mantenimiento de los sistemas de información tienen la obligación de velar que dicho sistema considerado como un activo de información se mantenga disponible, íntegro y confidencial, mientras esté en el proceso de desarrollo, mantenimiento y puesta en marcha.

- Todos los procesos que requieran cualquier tipo de desarrollo de software deberán dar cumplimiento de la política de desarrollo de software

2.3 REQUERIMIENTOS PARA DESARROLLO DE SOFTWARE

Todo requerimiento relacionado con Desarrollo de Software deberá ser tramitado por la Herramienta Aranda y soportado por el Formato de Solicitud de Requerimiento PA04-M02-F01, anexo al presente manual.

Adicionalmente el requerimiento de Desarrollo de Pruebas Sistemas de Información se deberá solicitar mediante el formato PA04-M02-F02 y para mayor claridad en el proceso del desarrollo interno está el formato PA04-M02-F03 de Especificaciones del Proceso de Desarrollo de Software y Flujograma anexo a este manual.

2.4 ARQUITECTURAS DE REFERENCIA PARA EL DESARROLLO DE SOFTWARE.

Todo proyecto de desarrollo de software en ambiente Web que se desee adelantar en la entidad, debe adoptarse bajo el modelo de capas o N-Capas[2].

En las capas se puede desacoplar la infraestructura de la lógica del negocio o la implementación de un servicio de la lógica de negocio.

Para una correcta implementación de la arquitectura se debe tener en cuenta las siguientes separaciones:

- **Dominio**: Contendrá la definición del objeto de dominio, los value object[3], las excepciones de dominio y la interfaz que va a interactuar con la infraestructura.
- **Aplicación**: Contendrá los casos de usos del dominio, en otras palabras, contendrá las acciones que podrá realizar ese objeto de dominio.
- **Infraestructura**: Contendrá la implementación de la interfaz definida en el dominio.

Se debe recordar que la capa superior podrá conocer las capas que se encuentren a un nivel inferior a ellas.

[2] *Estilo de programación donde el objetivo principal es separar los diferentes aspectos del desarrollo.*

[3] *Utilizados para presentar valores que no cambian con el tiempo y que no tienen un identificador único.*

2.5 ATRIBUTOS DE CALIDAD

La construcción de una buena arquitectura de software impacta positivamente sobre su capacidad de satisfacer las necesidades de las partes interesadas. Cada característica de está arquitectura (atributos de calidad) es una propiedad medible del sistema que permite evaluar aspectos de calidad tales como la disponibilidad, rendimiento, seguridad, usabilidad, escalabilidad, facilidad de pruebas, entre otros, como propuestos por la SDM para la construcción de todos sus productos de software.

De esta manera la entidad busca lograr de manera ordenada, estructurada, eficiente y segura, que sus aplicaciones y sus diferentes integraciones, minimicen los riesgos relacionados a la calidad, costos, tiempo y alcance; aumentando así la capacidad de aceptación del sistema por parte del usuario.

Disponibilidad: Antes de pasar un sistema a producción, cada proyecto de software deberá entregar un documento donde especifique los parámetros tenidos en cuenta para garantizar la disponibilidad del sistema.

Portabilidad: Toda aplicación desarrollada en la entidad debe tener la capacidad de cambiar de ambiente de producción de una manera ágil, sin afectar su funcionalidad inicial.

Rendimiento: Cada aplicación, hará un adecuado uso de los recursos ofrecidos para su entorno de producción de acuerdo con los que usa en condiciones específicas. También se requiere tener en cuenta el recurso utilizado en la interacción con otros productos de software para alguna funcionalidad en específico. De esta manera los tiempos de respuesta y procesamiento de la aplicación serán los apropiados.

Interoperabilidad: Todo desarrollo de software debe tener la capacidad de intercambiar datos bajo la tecnología REST[4] por medio del formato JSON[5], respetando los aspectos de seguridad establecidos en este documento.

Seguridad: Toda aplicación tendrá la capacidad de proteger la información y los datos de tal forma que los usuarios y/o sistemas no autorizados no puedan acceder a ellos. Para conceder al sistema los principios de integridad, autenticación y disponibilidad al sistema, se proponen los siguientes métodos:

Antes de realizar el paso a producción, todo proyecto de software debe contar con el visto bueno de la persona oficial de seguridad de la entidad o quien a su vez.

[4] *Interfaz entre sistemas que use HTTP para obtener datos o generar operaciones*

[5] *(JavaScript Object Notation) es un formato ligero de intercambio de datos*

- Es requisito para el paso a producción que los productos de software cuenten con un documento de análisis de riesgos como lo indica la guía de aseguramiento de aplicaciones.

- La persona oficial de seguridad deberá gestionar las vulnerabilidades en el desarrollo del software, revisando con una frecuencia mensual los logs generados por las aplicaciones o por el servidor de aplicaciones. Validar herramientas que permitan la automatización de este proceso.
- Se debe proteger la información sensible como (contraseñas, tokens, firmas digitales) en todos sus estados: tránsito, en reposo y en uso por medio de herramientas y técnicas criptográficas que garanticen la integridad y confidencialidad de la información.

- Las demás que indique el Manual de Políticas Específicas en Seguridad de la Información de la SDM – PA04-M01.

Usabilidad: Todos los sistemas y sus componentes desarrollados deben ser entendidos y aprendidos por el usuario de manera sencilla y práctica.

- Todo producto de software deberá seguir parámetros de usabilidad como simplicidad, organización de los elementos y adaptación a diferentes dispositivos, navegación intuitiva y accesibilidad sin importar su sistema operativo y/o navegador preferido.

- Cada componente desarrollado debe incluir documentación guía para el usuario, como lo son por ejemplo los video tutoriales, diagramas de proceso y manuales.

Escalabilidad: Cada aplicación debe poder manejar una carga constante y creciente de datos y trabajo, en simultaneo con el crecimiento de la Entidad. A fin de configurar el entorno de la aplicación para la atención de peticiones de manera conjunta y distribuir su carga de trabajo se establece:

- Se debe implementar un sistema de orquestador de contenedores por medio de Docker y Kubernetes[6] para garantizar el balanceo de cargas y el auto mantenimiento ante fallos por peticiones o no disponibilidad del sistema, contemplando el uso de firewall web o físico, lo anterior para garantizar la recuperabilidad del sistema y escalabilidad horizontal.

Facilidad de pruebas: Toda aplicación permitirá realizar pruebas a los componentes desarrollados o modificados para así poder identificar si provoca efectos secundarios o adversos durante su ejecución.

[6] *Docker Desktop se utiliza para ejecutar, editar y administrar el desarrollo de contenedores. Kubernetes se utiliza para ejecutar aplicaciones de producción a gran escala.*

A continuación, se expone cada uno de los tipos de pruebas propuestos para verificar y validar la aplicación:

Se deben realizar pruebas unitarias de código con la finalidad de garantizar el funcionamiento esperado de los métodos usados en el desarrollo.

Se aplicarán pruebas de integración sobre cada uno de los módulos y otros sistemas con los que se comunica la aplicación. De esta manera se podrá validar el comportamiento de los componentes presentes en el sistema y garantizar su funcionalidad.

Se deben realizar pruebas funcionales de sistema que validen su funcionamiento en condiciones específicas: sistemas operativos, navegadores, servidores, bases de datos o tipo de dispositivo en el que lo usen.

Posterior a una modificación y/o mejora de alguna funcionalidad en la aplicación, se realizarán pruebas de regresión para asegurar que estos cambios o adiciones no alteraron ni eliminaron funcionalidades existentes.

2.6 PRINCIPIOS Y PATRONES DE DISEÑO

- **Principios de Programación Orientada a Objetos**

El paradigma de programación orientada a objetos contiene los siguientes principios básicos:

- Abstracción
- Polimorfismo
- Herencia
- Encapsulamiento

- **Principios SOLID**

Los 5 principios SOLID de diseño de software son:

- ✓ S – Single Responsibility Principle o principio de responsabilidad simple.
- ✓ O– Open/Closed Principle o principio de abierto / cerrado.
- ✓ L – Liskov Substitution Principle o principio de sustitución de Liskov.
- ✓ I – Interface Segregation Principle o principio de segregación de interfaces
- ✓ D – Dependency Inversion Principle o principio de inversión de dependencias.

- Todo desarrollo de software debe seguir los principios de programación orientada a objetos que se alinean con los principios SOLID de forma obligatoria.

- Todo desarrollo de software debe implementar la arquitectura de software de referencia mencionada en esta política.

- Cada desarrollador de software debe implementar patrones de diseño en los proyectos en los que participe teniendo su alcance y propósito. Dado esto, se debe utilizar de acuerdo su clasificación: patrones creacionales, patrones estructurales y patrones de comportamiento.

- Cada proyecto de desarrollo debe incluir un ORM[7] para el mapeo de base de datos, validando que dicha herramienta no posea vulnerabilidades de inyección. Dado el caso en que no se pueda implementar el ORM en módulos específicos del proyecto, se deben realizar una justificación y utilizar sentencias preparadas para prevenir ataques de SQL[8] Injection.

2.7 METODOLOGÍA DE DESARROLLO

Durante el proceso de desarrollo de software se deben tener buenas prácticas y un proceso claro para trabajar colaborativamente en equipo. A la hora de poner en marcha un proyecto, toda empresa debe asegurar que el equipo implicado conoce las tareas y plazos de tiempo de entrega.

Scrum es una metodología de trabajo que nos ayuda a conseguirlo y que, además, permite agilizar la entrega de valor al cliente en iteraciones cortas de tiempo.

Esta etapa requiere generar los siguientes entregables, dando instrucción al procedimiento realizado para la puesta en producción de la aplicación:

Product Backlog (Artefacto): El Product Backlog es una lista emergente y ordenada de lo que se necesita para mejorar el producto. Es la única fuente del trabajo realizado por el Scrum Team.

Sprint Backlog (Artefacto): El Sprint Backlog se compone del Objetivo del Sprint (por qué), el conjunto de elementos del Product Backlog seleccionados para el Sprint (que), así como un plan de acción para entregar el Increment (como).

Se debe establecer para cada proyecto, módulo o formulario la metodología de gestión SCRUM en la cual se oriente el ciclo de vida de desarrollo en los proyectos de software de la SDM.

2.8 DOCUMENTACIÓN TÉCNICA

La OTIC debe disponer de un espacio en disco o plataforma (en este caso, se define como plataforma principal Azure DevOps) para que los equipos de desarrollo almacenen la información de documentación requerida.

[7] *Modelo de programación que permite mapear las estructuras de una base de datos relacional*

[8] *Lenguaje gestor para el manejo de la información en las bases de datos relacionales*

Todo desarrollo que se realice en la SDM debe generar los siguientes entregables:

- Plan de trabajo o Product Backlog
- Historias de usuario.
- Diagramas de clases, modelo de datos.
- Solicitud de cambios o ajustes a software (Documento Control de Cambios)
- Documento Plan de Pruebas.
- Guía de usuario.
- Códigos fuentes documentados.
- Paso a producción de software (Documento Control de Cambios)
- Diccionario De Modelo De Datos.
- Diagramas de arquitectura de software (componentes y despliegue)

2.9 LINEAMIENTOS Y ESTÁNDARES PARA EL DESARROLLO DE SOFTWARE

- Para el manejo de nomenclatura se deben adoptar estándares que aseguran un código legible, fácil de entender y mantener.

- Todo desarrollo de software debe especificar las tecnologías con las cuales se desarrollará los aplicativos, módulos o sistemas, y deben ser un estándar para los equipos de trabajos que realicen dichos desarrollos.

- Para garantizar seguridad de la información se deben establecer para cada proyecto la separación de ambientes de construcción, testing y producción cada uno representado en una rama del repositorio del proyecto, con el objetivo de controlar de mejor manera los cambios al sistema de información y lo demás que indique el Manual de Políticas específicas de Seguridad de la Información PA04-M01.

- Todo desarrollo de software debe contar con 3 ambientes: desarrollo, pruebas y producción. Cada uno con diferente base de datos.

- Todo desarrollo de software debe tener por lo menos 2 ramas en el repositorio de código, una para producción y una para desarrollo. Se recomienda tener una rama independiente para el ambiente de pruebas.

- Se deben implementar las pruebas de integración que sean necesarias a los sistemas con la finalidad de asegurar que en las aplicaciones desarrolladas se ha implementado los requisitos de seguridad definidos antes de comenzar el desarrollo.

- Se deben realizar pruebas de integración a las aplicaciones desarrolladas en conjunto con el/la dueño del sistema de información, haciendo uso del formato de casos de prueba, realizando la respectiva documentación a las pruebas realizadas y aprobando los pasos a producción

- Realizar el mergen de la rama de desarrollo a pruebas una vez finalizado el proceso de desarrollo y realizar el merge de la rama de pruebas a producción una vez validado y aprobado el proceso de pruebas integrales.

- Las herramientas de desarrollo de software utilizadas deben estar licenciadas por la Secretaría Distrital de Movilidad o el contratista según corresponda y deberá estar en custodia de la OTIC únicamente.

- Se debe hacer uso de un software de control de versiones el cual salvaguarde la integridad del código fuente desarrollado, la administración de los repositorios debe estar a cargo de una persona funcionaria y/o cuenta de la persona oficial de la SDM designada por la OTIC.

- Mantener actualizada las versiones de lenguaje de programación, frameworks[9] y librerías usadas en los proyectos de acuerdo con la tabla de versiones seguras, lo anterior para solventar los problemas de seguridad del lenguaje de las versiones anteriores.

- Se debe identificar frameworks y/o librerías y de fuentes reconocidos con mantenimiento, actualización y desarrollo activo por parte de la comunidad y que sean ampliamente usados en la industria. Los frameworks y/o librerías de terceros reconocidos que incluyen elementos de seguridad son primordiales para minimizar errores de implementación en componentes en los que el desarrollador no tenga el conocimiento y dominio necesario. Es de gran importancia realizar un inventario de dichos frameworks y librerías para tener un control de las actualizaciones y prevenir vulnerabilidad.

2.10 LINEAMIENTOS PARA EL DESARROLLO SEGURO

La política de desarrollo seguro de la SDM comprende las reglas para el desarrollo de software y sistemas dentro de la organización. Para esto, se establecen los siguientes lineamientos:

- Se deberán utilizar técnicas de programación seguras tanto para los desarrollos nuevos como en las situaciones de reutilización de códigos donde es posible que no se conozcan las normas que se aplican al desarrollo o donde no sean coherentes con las buenas prácticas actuales. Lo anterior, tanto para el desarrollo interno como externo.

[9] *Marco o esquema de trabajo generalmente utilizado por programadores.*

- Se deben estandarizar, los criterios de seguridad y calidad, que serán considerados, durante cada fase del proceso de desarrollo de sistemas de información.

- En cuanto al desarrollo de sistemas por terceros, se deben celebrar contratos, con las empresas proveedoras, que contengan cláusulas, que resguarden la propiedad intelectual para la SDM, y, asimismo, aseguren, los niveles de confidencialidad de la información, en el proyecto respectivo y lo demás que indique el Manual de Políticas específicas de Seguridad de la Información PA04-M01.

- Se debe diferenciar, entre la persona encargada de celebrar y autorizar los contratos con terceros, de los que deben validar y verificar su cumplimiento.

Para el atributo de seguridad, la SDM establece características específicas no funcionales como lineamientos a cumplir:

- **Captcha**: Se debe utilizar CATPCHA nivel 3 para las páginas de login, registro o contáctenos y otras que pidan información al usuario sin estar logeado en el sistema.

- **Seguridad API:** Se debe contar con un mecanismo de seguridad para el consumo del API basado en tokens. La autenticación es responsabilidad del API. Las autorizaciones a las diferentes funcionalidades del sistema son administradas desde el sistema mismo.

- **Cifrado de contraseña**: En el caso de que el sistema maneje contraseñas de usuario, estas deben ser encriptadas. Se debe definir el modo ECB o CBC, la longitud de bits debe ser 256, en base 64 utilizando una llave secreta (la cual debe ser parametrizable). El estándar de cifrado avanzado (AES) es un algoritmo de cifrado simétrico. AES es el estándar de la industria a partir de ahora, ya que permite el cifrado de 128 bits, 192 bits y 256 bits. El cifrado simétrico es muy rápido en comparación con el cifrado asimétrico y se utiliza en sistemas como el sistema de base de datos. A continuación, se muestra una herramienta en línea para generar una contraseña cifrada AES y descifrar una contraseña cifrada AES. Proporciona dos modos de cifrado y descifrado ECB y modo CBC.

- **Gestión de usuarios:** El sistema debe permitir la gestión de usuarios (creación, suministros de acceso, asignación de privilegios, revocatoria de accesos, etc.), roles y perfiles, grupos de usuarios, asociación de acciones para cada rol y administración exclusiva del administrador de la aplicación.

- **Auditoría del sistema**: Debe permitir generar registros de los ingresos a la aplicación y las actividades realizadas por los usuarios. Se debe utilizar la librería que se indique desde la OTIC.

- **Protección de datos personales:** El sistema debe garantizar el cumplimiento del régimen general de protección de datos personales. Cuando el sistema solicita información personal al usuario final, se debe establecer un mecanismo para obtener la autorización y tener prueba de ellos (log).

- **Cierre de sesión:** El sistema debe cerrar la sesión de trabajo luego de máximo 5 minutos de inactividad.

- **Características de contraseña:** El sistema debe exigir características especiales a las contraseñas: mínimo de 8 caracteres, símbolos, números, mayúsculas y minúsculas. Las características específicas de la estructura de la contraseña serán entregadas por la OTIC. La capa de presentación debe presentar una ayuda visual que indique cuales son las características que se requieren de la contraseña. Si la aplicación está integrada a un directorio activo, el directorio activo debe validar las características requeridas y aceptar o rechazar la contraseña

- **Bloqueo de contraseña:** Para aplicaciones que no están integradas con el directorio activo el sistema debe incluir controles de bloqueo de cuenta después de un máximo de 3 intentos erróneos a fin de evitar ataques de fuerza bruta. Si la aplicación está integrada al directorio activo este será el encargado de definir el número de intentos permitidos antes de bloquear la contraseña.

- **Vencimiento de contraseña:** Para sistemas que no estén integrados con el directorio activo el sistema debe controlar el vencimiento de la contraseña y solicitar al usuario cambio de la misma pasados 90 días, desde el último cambio de contraseña realizado.

- **Olvido de contraseña:** El sistema debe contar con opción de Olvido de contraseña, para esto se debe enviar por correo electrónico (a correo registrado por el usuario), link (solo valido por 24 horas) para reestablecer la contraseña.

- **Contraseña inicial:** El sistema debe detectar cuando es la primera vez de ingreso al sistema y solicitar cambio de contraseña.

- **Acceso redes públicas y privadas:** La aplicación debe cumplir con controles de seguridad relacionados a su utilización a través de redes públicas y privadas, garantizando la confidencialidad, integridad y disponibilidad de la información y acceso a ella. La OTIC indicara condiciones específicas a validar por parte del sistema.

- **Cifrado de comunicaciones:** Debe cifrar las comunicaciones de los servicios que puedan estar expuestos en redes públicas. El sistema debe permitir la implementación de certificados digitales. El sistema debe funcionar sobre protocolo TLS versión 1.2 mínimo (certificados internos de la entidad cuando los sistemas de información sean internos y certificados válidos públicamente cuando los sistemas de información estén expuestas a internet).

- **Análisis de vulnerabilidades:** Durante todo el proceso se realizarán análisis de vulnerabilidades que pueda tener el sistema. Para esto se establecerán puntos donde el software este adecuado para dicho análisis. en conjunto con la OTIC se analizarán las posibles vulnerabilidades y se determinaran los casos donde se deben realizar acciones de mitigación o cuales serán ignoradas. Para el paso a producción se debe realizar un último análisis de vulnerabilidades y validar que todo lo acordado de ajuste se tenga resuelto. Para esto la OTIC indicara herramienta, proceso y personal asignado para dicha actividad.

- **Manejo de JWT para seguridad de servicios:** Se debe utilizar JWT para la comunicación entre la capa de presentación y capa de servicios.

- **Manejo CORS:** Se debe validar que el API de servicios puede ser solo accedido por el dominio de la aplicación, sistemas, aplicaciones u otros servicios de la SDM

La SDM establece como buenas prácticas para el desarrollo seguro la adopción del estándar internacional OWASP (Open Web Application Security Project), cuyo propósito es mejorar la seguridad de los aplicativos Web. Se establecen como lineamientos técnicos las normas OWASP Top 10 que se mencionan a continuación:

Vulnerabilidad	Descripción	Controles
Inyección SQL	Esta vulnerabilidad se presenta cuando las consultas a la base de datos se construyen dinámicamente y las entradas de la aplicación no son controladas debidamente	• Pruebe y descarte fuertemente los caracteres especiales, espacios que no sean necesarios tanto del lado del cliente como del servidor • Para almacenar sus usuarios o al hacer la consulta verifique los meta caracteres SQL o LDAP, tanto del lado del cliente como del servidor Limite los caracteres del campo a la cantidad requerida • Use solo el nombre de usuario

Vulnerabilidad	Descripción	Controles
		como llave para las consultas • Verifique que solamente la consulta tengo uno o ningún resultado (si tiene más de uno es un error) • Limite el número de intentos de contraseña y tome medidas Si aplica use Certificados digitales
Cross-Site Scripting (XSS)	Esta vulnerabilidad se presenta cuando los datos de entrada se utilizan para construir el contenido de la aplicación sin ningún tipo de validación sobre éstos por medio de URL	• Pruebe y descarte fuertemente los caracteres especiales, espacios que no sean necesarios tanto del lado del cliente como del servidor • Verifique los datos de entrada y los de salida
Autenticación rota y administración de sesión	Esta vulnerabilidad se presenta debido a las fallas en la administración de las funciones de autenticación o sesión. Por ejemplo, exposición de usuarios, contraseñas y el ID de las sesiones	• Pruebe y descarte fuertemente los caracteres especiales, espacios que no sean necesarios tanto del lado del cliente como del servidor • Para almacenar sus usuarios o al hacer la consulta verifique los meta caracteres SQL o LDAP, tanto del lado del cliente como del servidor • Limite el número de intentos de contraseña y tome medidas • Si aplica, use Certificados digitales • Cifre la URL, si contendrá datos del usuario • No almacene los datos de los usuarios en las cookies. • Limite los tiempos de las sesiones

Vulnerabilidad	Descripción	Controles
Referencias directas a objetos de forma insegura	Se presenta cuando se expone una referencia a un objeto interno, como un archivo, directorio, registro de BD en la URL y no se implementa el debido control de acceso a los recursos	• Evite el uso del campo referencia ya que se puede modificar fácilmente. • Verifique un tipo valido de objeto relacionado • Si su aplicación tiene que usar la referencia, debería únicamente hacerlo como un mecanismo de defensa en profundidad • Si debe usar este tipo de referencias identifique los tipos de objetos y use métodos de uso para cada uno de estos
Cross-Site Request Forgery (CSFR)	Se presenta en aplicaciones donde las peticiones son fáciles de predecir y recrear	• Evite variables en las URL • Si debe usarlas, compruebe la información que contienen • No use la URL para enviar información sensible. • Identifique que los datos contenidos contengan el tipo de información que deberían contener • Cifre los datos de la URL • Use tokens como herramienta de validación • Valide sesiones
Pobre/Mala configuración de seguridad	Cuentas de acceso por defecto, directorios y archivos sin proteger, firewalls mal configurados, fallas sin parchar etc. Son las puertas utilizadas por los atacantes para ingresar	• No dejar la configuración de fabrica (default) • Asesórese de personal de infraestructura o en telecomunicaciones para verificar e implementar los

Vulnerabilidad	Descripción	Controles
	a la aplicación	requerimientos de la aplicación • Use fuentes oficiales para la descarga de la aplicación • Habilite solo los servicios estrictamente necesarios
Almacenamiento inseguro de Criptografía	Se presenta debido al pobre manejo de cifrado sobre los datos sensibles de la aplicación. Datos sin cifrar Algoritmos de cifrado pobres Almacenamiento inseguro de las llaves	• Use algoritmos de cifrado fuertes (AES, SHA-256). Evite DES, MD5 o SHA-1 • Si usa algoritmos asimétricos tenga en cuenta elementos de custodia segura para las llaves privadas • Si usa algoritmos simétricos tenga en cuenta la seguridad en la capad de transporte para la llave. (Diffie-Hellman para SSL) • Los algoritmos propietarios suelen usar “seguridad por oscuridad” lo que los hace muy inseguros, se recomienda algoritmos matemáticos. • Cifre solo los datos sensibles o que se defina en los requerimientos
Falla al restringir acceso por URL	Esta falla se presenta cuando las solicitudes a las páginas no se protegen adecuadamente. Los atacantes simplemente modifican la URL para tener acceso a páginas con privilegios que no están protegidas debidamente.	• Para detectar esta vulnerabilidad hay que identificar primero cuáles son las páginas que sólo son aptas para usuarios con altos privilegios. • Desde la cuenta con menos privilegios de la aplicación, modificar la URL para intentar acceder a las páginas que este usuario no puede acceder. Si se logra acceder la aplicación es

Vulnerabilidad	Descripción	Controles
		vulnerable.
Insuficiente protección de la Capa de Transporte	Si la información que atraviesa internet no está debidamente protegida, cualquier persona que monitoree la red puede obtener información sensible (Usuarios y contraseñas, ID de sesión).	Este tipo de vulnerabilidad está más relacionado con la infraestructura que con el aplicativo en sí. Se puede utilizar un sniffer para monitorear el tráfico de la red y detectar que información viaja sin protección
Redireccionamiento y reenvíos sin validación	Frecuentemente las aplicaciones redireccionan a los usuarios a otras páginas. Cuando esta se realiza por medio de parámetros que no se validan, el atacante puede escoger el sitio al que se redirecciona. Los atacantes pueden utilizar esta vulnerabilidad y engañar a los usuarios para que accedan a ciertos enlaces que redireccionan a páginas maliciosas. Esta vulnerabilidad también se utiliza para evadir controles de acceso	• Evite el uso del campo referencia ya que se puede modificar fácilmente. • Valide el contenido de campos de referencia como un analizador de registros Web, deben protegerse cuidadosamente contra XSS y otros ataques de inyección HTML. • Si su aplicación tiene que usar la referencia, debería únicamente hacerlo como un mecanismo de defensa en profundidad • Verifique privilegios en cada pagina • Por ejemplo, si login.jsp solo puede ser invocada desde http://www.example.com/index.jsp, la referencia podría verificar que la referencia sea este valor.

2.11 ESPECIFICACIONES PROCESO DEL DESARROLLO

Las especificaciones del proceso del desarrollo de software se encuentran definidos en el anexo PA04-M02-F03, por medio del cual se describe las fases de requerimientos, planeación, desarrollo y pruebas, revisión y controles; y despliegue.

3. ESTÁNDARES DE PROGRAMACIÓN

3.1 DISEÑO DE BASE DE DATOS

En la construcción de un software, no sólo se debe configurar las rutinas del programa para máximo desempeño, sino se debe tener en cuenta el diseño físico y lógico del almacenamiento de los datos.

- **Para un buen diseño de base de datos tener en cuenta:**

- Proveer un mínimo de tiempo de búsqueda para localizar registros específicos
- Almacenar los datos en la manera más eficiente posible
- Hacer las actualizaciones lo más fácil posible.
- Construcción la suficientemente flexible para la inclusión de nuevas funciones requeridas por el programa.

- **Objetivos del Diseño**

- Eliminar datos redundantes
- Poder localizar registros individuales rápidamente
- Realizar mejoras a la base de datos fáciles de implementar
- Fácil mantenimiento de la base de datos.

- **Actividades en el Diseño de la Base de Datos**

La creación de un buen diseño de la base de datos involucra las siguientes actividades:

- Modelamiento de la aplicación
- Determinar los datos requeridos por la aplicación
- Organizar los datos en tablas
- Establecer relaciones entre las tablas
- Usar índices y validaciones para los datos
- Crear y almacene cualquier búsqueda necesaria para la aplicación
- Revisar el diseño de la base de datos.

- **Organización de los Datos**

Uno de los aspectos importantes del diseño, es determinar cómo serán organizados los datos en la base de datos. Se deben organizar de manera que la información sea fácil de recuperar y realizar mantenimiento. Dentro de la base de datos, los datos son almacenados en una o más tablas. Se puede mantener un manejo eficiente de los datos almacenándolos en múltiples tablas y estableciendo relaciones entre ellas.

La normalización de datos es el proceso de eliminar datos redundantes dentro de una base de datos. Tomando la normalización en su concepto extremo, cada pieza de información en la base de datos debería aparecer una única vez, aunque esto no es siempre practico.

Una manera de manejar la normalización es lo que se conoce como tablas hijas. Las tablas hijas son tablas en las que todas las entradas comparten información común que es almacenada en otras tablas. Las tablas que contienen la información común se denominan tablas padres.

Se debe utilizar estos lineamientos:

- Determinar un tópico para cada tabla, y asegurar que todos los datos en la tabla se relacionan con el tópico
- Si un número de registros en la tabla contiene campos intencionalmente en blanco, dividir la tabla en dos tablas similares
- Si la información se repite en un número considerable de registros, mover la información a otra tabla y establecer relaciones entre las tablas
- Campos repetidos indican la necesidad de tablas hijas.
- Usar vistas para reducir el volumen de datos e incrementar la veracidad de los datos.
- No almacenar información en tablas si esta puede ser calculada de otras tablas.

- **Uso de Índices**

Cuando la información es ingresada a una tabla, los registros son almacenados en el orden en que fueron adicionados. Este es el orden físico de los datos. Sin embargo, se requiere un orden diferente al orden de entrada de los datos, este se define como orden lógico.

Un índice provee un método para mostrar la información de una tabla en un orden específico. Un índice es una tabla especial que contiene una llave (usualmente derivada de los valores de uno o más campos) para la búsqueda de datos en la tabla. El índice contiene un puntero que le dice al motor de la base de datos donde se encuentra un registro específico. Este tipo de índice es similar al índice de un libro.

La estructura de un índice permite localizar rápidamente los datos para recuperarlos. Si se tiene una tabla indexada alfabéticamente por el nombre, se puede encontrar un registro por un nombre específico rápidamente utilizando el índice.

Una tabla puede tener diferentes índices asociados dependiendo de la organización de la tabla. El tipo más común de índices son los índices de una sola llave, los cuales se basan en el valor de un solo campo de la tabla. Por ejemplo, la cédula, el NIT. Si se necesita más detalle de acuerdo a más campos de la tabla, se usan los índices de múltiple llave, que se basan en los valores de dos a más campos de la tabla.

- **Tablas**

Las tablas son objetos de la base de datos que contienen todos sus datos. Una definición de tabla consiste en una colección de columnas de la misma manera en que una base de datos es una colección de tablas. Para poder almacenar datos en una base de datos, primero es necesario comprender cómo se crean, modifican y mantienen las tablas de la base de datos. Esto incluye tareas como definir claves y agregar o eliminar columnas de tablas.

- **Índices**

Son elementos que se utilizan para mejorar el rendimiento de la aplicación de bases de datos, aumentan la velocidad de las consultas. Los índices se colocan sobre un conjunto de columnas en una determinada tabla, con el fin de realizar consultas ágiles.

Existen varios tipos de índices, dependiendo su finalidad:

- Únicos – Sin Duplicados
- De llave Primaria
- Con Duplicados.

- **Llaves Primarias**

Son restricciones y/o índices que identifican de manera única un registro en una tabla. La llave primaria está conformada por una o más columnas de una tabla. El ejemplo clásico de una llave primaria es el número de identificación en una tabla de clientes.

- **Llaves Foráneas – Relaciones**

Las diferentes llaves primarias e índices de una tabla pueden relacionar o estar relacionada en otra tabla, es decir que existen restricciones que hacen posible que dos tablas se puedan cruzar con efectividad, obtener datos con integridad al utilizar estas restricciones e índices.

- **Desencadenadores**

Objetos de la base de datos que se activan cuando una tabla los contiene, dependiendo de la acción que se ejecute, es decir existen varios tipos de desencadenadores:

- De inserción. Se activan cuando se ingresa un nuevo registro en la base de datos.
- De Borrado. Ejecuta el código específico cuando se eliminan datos de la base de datos.
- De Actualización. Este tipo de desencadenador puede apuntar a la actualización de cada una de las columnas de la tabla o a una columna específica para ser activado.

- **Procedimientos Almacenados**

Conjunto de instrucciones que pueden modificar o generar un resultado sobre los objetos propietarios de la base de datos. Los procedimientos almacenados (SP) pueden contener diferentes tipos de parámetros, cada uno con diferentes características:

Parámetros de Entrada: Variables que son valoradas desde otro procedimiento o aplicativo externo, pero que no son modificables en el mismo procedimiento.

Parámetros de Salida: Variables definidas en el encabezado del procedimiento, toman valor únicamente en el mismo procedimiento, para ser utilizadas en otros procedimientos o aplicativos externos.

Parámetros de Entrada – Salida: Son valoradas por otros procedimientos o aplicativos y a la vez pueden ser modificadas y externalizadas por el mismo procedimiento que las contiene.

- **Vistas**

Son consultas que se guardan en la base de datos y que son utilizadas por otros objetos. Su utilización en la generación de datos es similar a la de una tabla en la base de datos.

- **Funciones**

Tiene la misma funcionalidad de un procedimiento almacenado, con la salvedad que puede ser utilizado en una vista, pues devuelve un tipo de dato específico.

3.2 ESTÁNDARES PARA BASES DE DATOS

TABLAS	**Las tablas se nombrarán de acuerdo con la función que desempeñe en la base de datos. La estructura es la siguiente:** ***XXX_NOMBRE_TABLA** ***XX: Nombre del Proyecto** ***X: Tipo de Tabla** ***Nombre: Finalidad de la tabla.** **La restricción de longitud del nombre debe ser máximo de 50 caracteres, sobra decir que debe ser muy diciente el nombre de la tabla, y en lo posible en las herramientas que contiene comentarios utilizarlos. Los tipos de tabla son los siguientes:** ***B: Tablas Básicas** ***H: Tablas Históricas** ***I: Tablas de Interfaces** ***T: Talas temporales** **Ej: MCH_CLIENTE** **Los comentarios en las tablas y en los campos son obligatorios. Los motores de bases de datos tienen opciones o comandos para comentar los objetos de la base de datos.**
INDICES	Se rigen de acuerdo a la siguiente estructura: IDX_NOMBRE_INDICE El nombre del índice debe contener el nombre de la tabla a la cual hace referencia, y utilizar los comentarios asociados, si los tiene la aplicación para describir su funcionamiento. Si existe más de un índice asociado, enumerarlo en orden secuencial al último creado Ej: IDX_MCB_PARAMETRO.
LLAVES PRIMARIAS	Deben tener la siguiente estructura PK_NOMBRE_TABLA La llave primaria debe contener el nombre de la tabla dentro de su estructura, la combinación pk determina que la restricción o el índice cumplen con el condicionamiento de unicidad. Ej: PK_MCB_PARAMETRO
LLAVES FORÁNEAS	Las relaciones de integridad deben denominarse de la siguiente forma: FK_TABLAREFERIDA_TABLAREFERENCIA El nombre de la referencia debe contener los nombres de las dos tablas que la conforman. Se hace referencia a la tabla referida como aquella que tiene la restricción, y la tabla de referencia como aquella de donde se toman los datos que van a ser comparados. Ej: FK_MCH_DETALLE_MCH_CLIENTE01. El consecutivo es para diferenciar la coexistencia de más de una relación entre las dos tablas.
STORED PROCEDURE o PROCEDIMIENTOS ALMACENADOS	Su estructura es la siguiente: XXSP_NOMBRE_PROCEDIMIENTO XX: Iniciales del proyecto El nombre del procedimiento debe ser acorde con la función que desempeñe. Ej: MCSP_CALCULARFORMULA

VISTAS	La denominación de la vista desde ser: XXV_NOMBRE_VISTA XX: Iniciales del proyecto. NombreVista: Debe ser claro, describiendo la función que realiza. Ej: MCV_CLIENTE_ACTIVO La nomenclatura también aplica para el objeto VISTA MATERIALIZADA de ORACLE
FUNCIONES	Están compuestas por: XXSF_NOMBRE_FUNCIO N XX: Iniciales del Proyecto Nombre Funcion: Debe ser claro, describiendo las operaciones que ejecuta. Ej: MCSF_NOMBRE_CLIENTE.
RESTRICCIONES DE CHEQUEO	En caso que se requieran incluir restricciones de chequeo, estas deben contener el nombre de la tabla, seguido de “CHK” seguido de al menos el nombre de una de las columnas que lo componen. Ej: CHK_MCH_CLIENTE_TIPO_ENTIDAD
PAQUETES	Para el caso de las bases de datos que utilicen paquetes como ORACLE, se debe utilizar el siguiente estándar XXPQ_NOMBREPAQUETE XX: Perefijo de la base de datos PQ: Identifica que se trata de un paquete Ejemplo: MCPQ_GENERAR_CONSULTA Los procedimientos y/o funciones dentro de los paquetes se nombran con el prefijo SP o SF según sea el caso. Ejemplo: MCPQ_RETENCION.SP_CALCULAR
COMENTARIOS	Después del encabezado del procedimiento o función se debe registrar la información correspondiente al objetivo del procedimiento, fecha de creación, parámetros de entrada y salida, así: ---------- -- DESCRIPCION: Este procedimiento calcula los totales de las notas credito activas y las registra en TCAR_CLIENTES -- PARAMETROS: Descripción clara de los parametros de entrada y salida. -- MODIFICACIONES: JUAN PEREZ FECHA CREACION: 01/01/2023 FINALIDAD DE LA MODIFICACION ---------- Se deben incluir todos los comentarios y explicaciones que se consideren para hacer el código claro de entender, estos de deben realizar empleando doble guión antes del comentario.

TRIGGERS	Debe estructurarse de la siguiente forma: TRG_NOMBRE_TABLA TRG: Referido a un desencadenador NombreTabla: Es el nombre sobre la tabla en que actúa el desencadenador. Ej: TRG_MCH_FACTURA01 El consecutivo es para diferenciar la coexistencia de más de un desencadenador del mismo tipo en la misma tabla. En el cuerpo del trigger debe incluirse un área para los comentarios o descripción de la finalidad del mismo.
SECUENCIAS	Las secuencias aplican para base de datos ORACLE, y se usan como generadores de numeros secuenciales. La estructura es XXSQ_NOMBRE_SECUENCIA Ejm: MCSQ_CONSECUTIVO_DETALLE
VARIABLES DENTRO DE PROCEDIMIENTOS Y FUNCIONES	Independientemente del motor de base de datos, las variables utilizadas dentro de la funcionalidad de los procedimientos de nombran de la siguiente manera. Para los nombres de las variables utilizadas dentro del cuerpo del procedimiento. V_NOMBRE_VARIABLE Ejms: V_CEDULA_CLIENTE NVARCHAR (15) SQLSERVER V_CEDULA_CLIENTE VARCHAR2(15) ORACLE Para los nombres de los parámetros que son recibidos en los procedimientos o funciones P_NOMBRE_PARAMETRO Ejms: P_CODIGO_CUIDAD NUMERIC (5) SQLSERVER P_CODIGO_CIUDAD NUMBER (5) ORACLE Para la definición de los nombres de los cursores. C_NOMBRE_CURSOR Ejms: DECLARE C_NORMAS CURSOR FOR SELECT …. SQLSERVER
RECOMENDACIONES	Utilizar la directriz %ROWTYPE, para definir las variables del mismo tipo del campo definido en la tabla. Ejm: V_NOMBRE_CLIENTE%ROWTYPE (EXCLUSIVO DE ORACLE) Utilizar objetos de tipo DOMINIO, que identifican tipos de datos definidos por el usuario. Aplicable para SQLSERVER

3.3 ESTÁNDARES DE PROGRAMACIÓN PARA JAVA

El documento pretende continuar con estándares internacionalmente aceptados y desarrollados en los diferentes entornos de desarrollo (IDE's)

Se puede consultar la referencia los estándares en:
https://www.oracle.com/java/technologies/javase/codeconventions-contents.html

- **Comentarios iniciales**

La programación en lenguaje JAVA, implica la programación de clases, en cada uno de los archivos que contienen las clases se analizará la estructura del mismo en orden descendente.

Ejemplo:

```
/*
* Nombre de la clase:
*
* Versión:
*
* Fecha:
*
* Derechos de Autor:
*/
```

- **Sentencias de Package e Import**

El paquete donde se encuentra programando debe ser en orden inverso de cómo se llamaría en su página web, Por ejemplo:

```
package com.softmanagement.estandares;
import java.awt.peer.CanvasPeer;
```

- **Declaraciones de clase e Interfases**

En la tabla siguiente se describen las diferentes partes de la declaración de una clase o interfase, y en el orden que deben aparecer:

Declaración	Comentarios
Comentario de documentación (/**...*/)	En la parte de Comentario de Documentación se indica la información que debe ir en el comentario de documentación del javadoc el cuál debe contener el motivo de la creación de la clase.

Declaración	Comentarios
Sentencia class o interface	
(SUGERENCIA) Comentarios a la implementación de la clase/interface (/*...*/), en caso necesario	Este comentario debería contener cualquier información de alcance que no sea adecuada para estar situada en el comentario que permite generar posteriormente la documentación.
Variables de la clase (estáticas)	En primer lugar las variables public, luego las protected, luego las variables de paquete (sin modificador de acceso) y finalmente las private.
Variables de instancia	Primero las públicas, luego las protegidas, luego las variables de paquete (sin modificador de acceso) y las privadas.
Constructores	
Métodos	Estos métodos deben estar agrupados por funcionalidad, en vez de por el alcance o accesibilidad. Por ejemplo, un método de una clase privada puede encontrarse entre dos métodos de instancia públicos. La finalidad es que el código sea más fácil de leer y entender.

- **Indentación**

Deben tomarse 4 espacios como unidad de indentación. La construcción exacta de la indentación (espacios o tabuladores) no es crítica.

- **Corte de líneas**

Cuando una expresión no cabe en una sola línea, debe saltarse a la siguiente línea de acuerdo con los siguientes principios generales:

- Saltar después de una coma
- Saltar antes de un operador
- Son preferibles los saltos de alto nivel a los de bajo nivel
- Alinea la nueva línea con el inicio de la expresión del mismo nivel de la línea inmediatamente anterior

Las líneas de código siguientes muestran tres ejemplos aceptables de formateo de expresiones ternarias.

```
alpha = (unaExpresionBooleanaMuyLarga) ? beta : gamma;
alpha = (unaExpresionBooleanaMuyLarga) ? beta
        : gamma;
alpha = (unaExpresionBooleanaMuyLarga)
```

```
    ? beta
    : gamma;
```

- **Espaciado**

Los espacios en blanco y líneas vacías proporcionan mayor legibilidad al código que se escribe, tanto porque las sentencias se visualizan mucho más claramente como porque las secciones de código diferentes se encuentran separadas adecuadamente.

- **Líneas en blanco**

Se debe utilizar siempre una línea en blanco de separación en las siguientes circunstancias:

- Entre métodos
- Entre las variables locales en un método y su primera sentencia
- Antes de un bloque de código o un comentario de una línea
- Entre secciones lógicas dentro de un método para incrementar la legibilidad del código

Es conveniente el uso de dos líneas en blanco en las circunstancias que se indican:

- Entre secciones dentro de un archivo fuente
- Entre las definiciones de clases e interfaces

- **Convenciones de nomenclatura**

Las convenciones de nomenclatura hacen que los programas sean más estándar y fáciles de leer, ya que cualquier programador está acostumbrado a tratar con código escrito de forma semejante. Además, proporcionan información sobre la funcionalidad del identificador; por ejemplo, si es una constante, un paquete o una clase, lo que también redunda en una ayuda adicional a la hora de entender el código.

Identificador	Reglas de Nomenclatura	Ejemplo
Clases	Los nombres de clases deben ser palabras completas, en mayúsculas y minúsculas, con la primera letra de cada palabra en mayúscula. Los nombres de clases deben ser simples y descriptivos, utilizando palabras completas y acrónimos o abreviaturas (a no ser que la abreviatura sea ampliamente conocida, como URL o HTML).	class Usuario; class ImagenLarga;

Identificador	Reglas de Nomenclatura	Ejemplo
Interfaces	Los nombres de interfaces deben seguir las mismas reglas indicadas para las clases, además debe tener el prefijo I.	interface IDelegado; interface IAlmacenado;
Excepciones	Los nombres de las Excepciones deben seguir las mismas reglas indicadas para las clases, además debe tener el sufijo Exception	class ErrorDatosException

Métodos	Los métodos deberían ser verbos, en mayúsculas y minúsculas, con la primera letra en minúscula, y la primera letra de cada una de las palabras internas en mayúscula. Para los métodos Bean los métodos deben empezar con get y set.	correr();
		correrRapido();
		getColor();

Variables	Todos los nombres de variables de instancia o de clase deben estar constituidos por palabras con la primera letra de la primera palabra en minúscula y la primera letra de las palabras internas en mayúscula. Los nombres de variables deben ser cortos y significativos. La elección de un nombre de variable debe ser mnemotécnico, es decir, pensado para que un lector casual al verla comprenda su uso. Se deben evitar las variables de una sola letra, excepto en variables temporales de corto uso. Nombres comunes para este tipo de variables son: i, j, k, m y n para enteros; c, d, y e para caracteres. Es aceptado el uso del subrayado como prefijo (“_”) en las variables privadas de la clase.	int i; float miAncho;
Constantes	Los nombres de variables declaradas como constantes de clase y constantes ANSI, deberías escribirse siempre en mayúsculas, con las palabras internas separadas por el signo de subrayado ("_"). Las constantes ANSI se deben evitar en lo posible, para facilitar la depuración del código.	static final int ANCHO_MINIMO = 4; static final int ANCHO_MAXIMO = 999;

- **Técnicas de programación**

Como norma general, se debe programar en ESPAÑOL, es decir se debe evitar los nombres en inglés a no ser que sea absolutamente necesario, que su traducción traiga consigo una confusión, o que en el entorno del negocio se facilite la nominación. Estas son algunas sugerencias acerca de cuestiones de programación más habituales y comunes.

- **Proporcionar acceso a variables de clase e instancia**

No hacer nunca una instancia o una variable de clase pública sin una buena razón. Normalmente, las variables de instancia necesitan que sus valores sean fijados o recogidos explícitamente; a menudo esto es un efecto lateral de la llamada al método. Un ejemplo de uso adecuado de las variables de instancia públicas es el caso en que la clase es esencialmente una estructura de datos, sin ninguna funcionalidad

- **Referencias variables y métodos de clase**

Evitar el uso de un objeto para acceder a una variable de clase (estática) o método. Utilizar el nombre de la clase en su lugar. Por ejemplo:

```
metodoDeClase(); // OK
UnaClase.metodoDeClase(); // OK
unObjeto.metodoDeClase(); // EVITARLO!
```

- **Asignación de variables**

Las constantes numéricas (literales) no deben codificarse directamente, excepto para –1, 0 y 1; porque pueden aparecer en un bucle for como valores de contador.

- **Paréntesis**

Es siempre una buena idea el uso abundante de paréntesis en expresiones que involucren a varios operadores para evitar los problemas generados por la precedencia de operadores. Incluso aunque la precedencia de los operadores parezca clara al programador, es posible que no lo sea tanto para otros, por lo que se debe asumir que esa precedencia no es bien conocida por todo el mundo. Y además, los paréntesis son gratis, así que pueden usarse libremente.

```
if (a == b && c == d) // EVITARLO!
if ((a == b) && (c == d)) // CORRECTO
```

- **Retorno de valores**

```
if (expresionBooleana) {
   return true;
} else {
  return false;
}
```

debería escribirse de la siguiente forma:

```
return expresionBooleana;
```

De forma semejante,

```
if (condicion) {
return x;
}
return y;
```

debería escribirse de la siguiente forma:

```
return (condicion ? x : y);
```

- **¿Expresión antes del? de una condición**

¿Si una expresión conteniendo un operador binario se coloca antes del interrogante en una expresión ternaria del tipo ?:
Debería colocarse entre paréntesis.
Por ejemplo: (x >= 0) ? x : -x

3.4 ESTÁNDARES DE PROGRAMACIÓN PARA .NET

Variable	Las variables son estructuras de datos que, como su nombre indica, pueden cambiar de contenido a lo largo de la ejecución de un programa. Una variable corresponde a un área reservada en la memoria principal del computador pudiendo ser de longitud: ● Fija, cuando el tamaño de la misma no variará a lo largo de la ejecución del programa. Todas las variables, sean del tipo que sean tienen longitud fija, salvo algunas excepciones — como las colecciones de otras variables (arrays) o las cadenas. ● Variable, cuando el tamaño de la misma puede variar a lo largo de la ejecución. Típicamente colecciones de datos.
Propiedades	Las propiedades son miembros que ofrecen un mecanismo flexible para leer, escribir o calcular los valores de campos privados. Se pueden utilizar las propiedades como si fuesen miembros de datos públicos, aunque en realidad son métodos especiales denominados descriptores de acceso. De este modo, se puede tener acceso a los datos con facilidad, a la vez que

	proporciona la seguridad y flexibilidad de los métodos.
Clase	Las clases son declaraciones o abstracciones de objetos, lo que significa, que una clase es la definición de un objeto. Cuando se programa un objeto y se definen sus características y funcionalidades, realmente se programa una clase. Tipo de referencia que encapsula datos (constantes y campos) y el comportamiento (métodos, propiedades, indizadores, eventos, operadores, constructores de instancia, constructores estáticos y destructores), y puede contener tipos anidados. Los tipos de clase admiten la herencia, un mecanismo mediante el cual una clase derivada puede extender y especializar una clase base.
NameSpace	Namespace o espacio de nombre simplemente es un conjunto de nombres en el cual todos los nombres son únicos. Un namespace es un contexto en el que un grupo de uno o más identificadores pueden existir. Un identificador definido en un namespace está asociado con ese namespace. El mismo identificador puede independientemente ser definido en múltiples namespaces, eso es, el sentido asociado con un identificador definido en un namespace es independiente del mismo identificador declarado en otro namespace. Los lenguajes que manejan namespaces especifican las reglas que determinan a qué namespace pertenece una instancia de un identificador. Generalmente se nombrarán en Plural, auque en algunas ocasiones se utilizará el singular por ejemplo cuando éste tenga un Nombre Corto de 2 o 3 caracteres por ejemplo IO. Se deberá escribir en Mayúscula su primer Carácter, o primer carácter de cada Palabra si éste es compuesto. Algunos ejemplos: ● SM ● Administraciones ● AdministracionUsuarios.
Clases	● Se llamarán sin algún tipo de excepción se manera Singular. ● Su nombre no contendrá Prefijos. ● Se deberá escribir en Mayúscula su primer Carácter, o primer carácter de cada Palabra si ésta es compuesta. ● En la medida de lo posible tendrán regiones descriptivas para una mejor navegación en ella. ● Ninguna se debe llamar como se llama su contenedor o NameSpace (SM.Sm). ● En la medida de lo posible se protegerá la clase si su funcionalidad definitivamente no se heredará.

Interfaces	● Se llamarán sin algún tipo de excepción se manera Singular. ● Su nombre Iniciará con el Prefijo I. ● Se deberá escribir en Mayúscula su primer Caracter, o primer caracter de cada Palabra si ésta es compuesta. ● Ninguna se debe llamar como se llama su contenedor o NameSpace (IO.Io). Ejemplo: ● IAdministracionUsuario.
Variables de Clase	Toda variable de este tipo debe ser Generalmente Privada. Después de su Prefijo debe iniciar con minúscula, si es palabra compuesta la primera letra de la palabra siguiente iniciará con Mayúscula. Ejemplo: _variablesCompuesta.
Propiedades	Toda variable de este tipo debe ser Privada/Publica. Propiedad comienza en Mayúscula. Ejemplo: VariableCompuesta.
Estructuras	Serán de Tipo Publico. Se llamarán en Mayúscula/Plural Y sus tipos en Minuscula/Singular. Nota: Código Auto generado por .Net no se debe tocar bajo ninguna circunstancia.
Métodos	● Ninguno se debe llamar como se llama su contenedor. ● Empieza con Letra Mayúscula. ● La Primera palabra del Método será un verbo. Ejemplo: Guardar.

4. PLATAFORMA TECNOLÓGICA

Los sistemas deberán estar desarrollados sobre plataforma WEB utilizando las siguientes tecnologías, herramientas y software base:

Componente Tecnológico	Descripción
Sistemas Operativos de Servidores	Linux versiones instaladas o más recientes Windows Server versiones instaladas o más recientes
Base de datos	Oracle 11G o versión más reciente SQL Server versiones instaladas o SQL Azure
Servidor WEB	Apache Web Server versiones instaladas o más recientes Apache Tomcat versiones instaladas o más recientes
Entorno de desarrollo	Visual Studio Code Visual Studio .Net

	IntelliJ IDEA .NET Framework 4.5 o versiones más recientes Java JDK 8 o versiones más recientes
Lenguaje de programación	Java, C#, JavaScript
Navegador Web (Browser)	Google Chrome versión más reciente Microsoft Edge versión más reciente

5. ESTRUCTURACIÓN DE SOLUCIONES Y PROYECTOS

Las siguientes son las reglas aplicables a los desarrollos nuevos y mantenimientos sobre componentes de software:

- **Nombres de los proyectos**

Para definir los nombres de los proyectos que hacen parte de una solución se debe tener en cuenta los siguientes ítems:

Se denomina como “Solución principal”, a un conjunto ordenado de proyectos que son integrados y que funcionan para entregar un sistema de información; la solución puede contener múltiples proyectos, servicios, bibliotecas, componentes, páginas WEB, archivos de configuración, sitios WEB, entre otros.

La Solución principal debe permitir que cada uno de los componentes que hacen parte de él tenga independencia funcional de los demás componentes del sistema, por lo que para facilitar el desarrollo independiente se crea una solución que contiene los proyectos que conformen el componente, el nombramiento de estas soluciones debe ser de la siguiente forma: XXX.Componente.

XXX: Corresponde a la abreviatura del sistema de información que se está desarrollado, por ejemplo: SAP.Seguridad.

Para los casos de capas de desarrollo o componentes de nivel jerárquico más alto, por ejemplo: capa de presentación, capa de servicios, capa de negocio, entre otras, el nombramiento considera el nombre lógico de la capa a utilizar, de la siguiente forma: XXX.Capa.Componente

Para el sistema SAP, capa de presentación WEB y componente de seguridad, quedará de la siguiente forma: SAP.Web.Seguridad

- **Estructura**

Los proyectos que se construyen en la solución principal, deben cumplir con la siguiente estructura, dependiendo de la funcionalidad que cumple y a que capa del sistema de información está afectando: XXX.Componente.CapaOFuncionalidad

Para el sistema SAP, componente de seguridad y capa de datos es: SAP.Seguridad.Datos

En los casos que dentro de una misma solución deban existir más de un proyecto que haga referencia a una capa del sistema, como la capa de presentación y entidades por hacer referencia a modelos de base de datos diferente, la estructura de estos proyectos debe incluir el nombre del modelo que se está afectando. XXX.Componente.CapaOFuncionalidad.Modelo

Para el sistema SAP, componente de WorkFlow, capa de datos, en la entidad de datos Factura, quedará de la siguiente forma: SAP.Workflow.Datos.Factura, o SAP.Workflow.Entidades. Factura

Para cada solución se recomienda la utilización de directorios para cada tipo de proyecto. Un ejemplo de esto es que se pueden tener varios proyectos en una solución con tecnologías diferentes por ejemplo proyectos web ASP.NET, proyectos de WinForms así como WPF, WCF, Web Services etc. De esta manera se pueden tener varios proyectos y varios folders para cada proyecto enmarcado en la tecnología que utiliza.

- /Codigo
- /WinForms
- /Web
- /WCF
- /LibreriadeClases1
- /LibreriadeClases2
- SolucionPrincipal.sln