Contenido

1. Introducción 5

2. Principios Generales 5

3. Arquitectura del Sistema 5

4. Estándares de Codificación 7

4.1. Lenguajes de Programación 7

4.1.1. JAVA: 7

4.1.2. .NET: 9

4.1.3. Frontend 10

5. Convenios de Nomenclatura 14

5.1. JAVA 14

5.2. C#: 16

5.3. Servicios Web (Rest) 19

5.3.1. Estructura base 19

5.3.2. Convención de nombres 20

5.3.3. Acciones permitidas 20

5.3.4. Estándar CRUD corporativo 20

5.3.5. Estándar de Respuesta para Servicios 24

6. Versionamiento de Aplicaciones 28

6.1. Versionamiento semántico 28

6.2. Versionamiento en Front-End 30

6.2. Versionamiento en Back-End 31

6.3. Versionamiento de Base de Datos 32

6.4. Consistencia entre componentes 33

6.5. Metadato description 34

6.6. Requisito de cumplimiento 34

7. Mensajería 35

7.1. Estructura del nombramiento 35

7.2. Tipos de colas 35

7.3. Tipos de Exchanges 36

8. Lineamiento para la documentación del proyecto – README.md 36

8.1. Introducción 37

8.2. Características del proyecto 37

8.3. Tecnologías y versiones 37

8.4. Prerrequisitos 38

8.5. Obtener el código fuente 38

8.6. Compilación del proyecto 38

8.7. Despliegue 38

8.8. Ejecución de la aplicación 39

8.9. Documentación de servicios – Swagger/OpenAPI 39

8.10. URL de los ambientes 39

8.11. Configuración adicional 40

8.12. Estructura mínima obligatoria del README.md 40

8.13. Criterios de cumplimiento 41

9. Lineamientos de Registro de Logs para Aplicaciones y Microservicios 42

9.1. Objetivo 42

9.2. Configuración General 42

10. Lineamientos de Auditoría para Aplicaciones y Microservicios 47

10.1. Objetivo 47

10.2. Alcance 47

10.3. Operaciones Permitidas 48

10.4. Reglas para el Manejo de Contenidos 52

11. Estructura del Código 53

11.1. Control de Versiones 54

11.2. Repositorio 55

11.3. Ramas 55

11.4. Commits 55

12. Base de Datos 59

12.1. Nombre de la Base de Datos 59

12.2. Nombre de las instancias de Bases de Datos 60

12.3. Nombre de esquemas 60

12.4. Convención General 62

13. Pruebas 70

13.1. Tipos de Pruebas 70

13.2. Cobertura de Pruebas 71

13.3. Herramientas de Pruebas 71

14. Entorno de Desarrollo 72

14.1. Configuración 72

14.1.1. Java 72

14.1.2. C# 73

14.2. Requisitos para desarrollar 74

14.2.1. Java 74

14.2.2. C# 75

14.3. Entornos 77

14.4. Herramientas 77

14.4.1. Java (Editores de código recomendados) 77

14.4.2. C# (Editores de código recomendados) 78

14.5. Herramientas de Integración 79

15. Seguridad 79

15.1. Normas para Manejar Credenciales y Permisos 80

15.2. Manejo de Datos Sensibles 81

15.3. Cabeceras en aplicaciones web 81

15.4. Uso de la Inteligencia artificial en el desarrollo 84

15.4.1. Mentalidad Fundamental: "Engineer-in-the-Loop" 84

15.4.2. Generar Código: Se debe especificar y controlar el proceso 85

15.4.3. Seguridad de Datos y Código 85

15.4.4. Manejo de Errores y Monitoreo 86

15.5. Consideraciones de Propiedad Intelectual (Derechos de autor) 86

15.5.1. Propósito y Alcance 86

15.5.2. Principios Fundamentales 86

15.5.3. Prohibiciones Explícitas 87

15.5.4. Usos Permitidos y Buenas Prácticas 88

15.5.5. Procedimiento de Autorización y Uso Seguro 89

15.5.6. Responsabilidades 89

15.5.7. Infracciones y Sanciones 90

15.6. Revisión de Seguridad 90

16. CONTROL DE CAMBIOS 91

1. Introducción

Este documento establece los lineamientos y mejores prácticas para el ciclo de desarrollo de software para la superintendencia de sociedades. Cuyo propósito es garantizar la calidad, consistencia y mantenibilidad del código.

2. Principios Generales

- Código limpio y mantenible.
- Adherencia a los estándares de la industria.
- Facilidad para pruebas y despliegue continuo.
- Colaboración efectiva entre desarrolladores.
- Mejorar en tiempos de mantenimiento de las diferentes aplicaciones.
- Calidad en la aplicación entregada, afectando positivamente la experiencia del usuario.
- Minimizar riesgos sobre perdida de información y/o vulnerabilidades de los diferentes sistemas de información expuestos.
- Estandarización sobre el ciclo de desarrollo de software para los contratistas como para los diferentes proveedores de la superintendencia de sociedades.

3. Arquitectura del Sistema

Los diagramas de arquitectura sirven para representar visualmente el diseño y la estructura de sistemas o aplicaciones, facilitando la comprensión, comunicación y planificación de proyectos tecnológicos. Estas son sus principales funciones:

- **Comunicación entre equipos**: Los diagramas permiten que diferentes

equipos, como desarrolladores, arquitectos de software, gerentes de proyecto y clientes, entiendan cómo interactúan los componentes de un sistema. Esto es fundamental para alinear expectativas y reducir malentendidos.

- **Planificación y diseño**: Ayudan a los arquitectos de software a diseñar la estructura de aplicaciones o sistemas, decidiendo qué componentes se necesitan, cómo se conectan entre sí y qué tecnología utilizar en cada caso.
- **Detección de problemas y optimización**: Permiten identificar puntos débiles o cuellos de botella en el diseño, lo que facilita la optimización del sistema antes de su desarrollo o implementación.
- **Documentación**: Los diagramas actúan como una guía visual de referencia que documenta la estructura del sistema, facilitando el mantenimiento y futuras ampliaciones.
- **Análisis de dependencias**: Ayudan a visualizar cómo los diferentes componentes, módulos o servicios interactúan y dependen unos de otros, lo cual es crucial para entender el impacto de cambios en el sistema.

Los diagramas que debe generarse en el ciclo de desarrollo son:

- **Realizar Diagrama de arquitectura** (si aplica): Debe ser realizado por el líder técnico y/o arquitecto de software.
- **Realizar Diagrama de componentes** (si aplica): Debe ser realizado por el líder técnico y/o arquitecto de software.
- **Realizar Diagrama de clases o de dominio** (si aplica): Debe ser el desarrollador de software.
- **Realizar Diagrama de secuencia** (si aplica): Debe ser el desarrollador de software.
- **Realizar Diagramas de flujo**: Debe ser el desarrollador de software.
- **Definición de la especificación de las maquinas**

4. Estándares de Codificación

4.1. Lenguajes de Programación

Los lenguajes de programación autorizados para el desarrollo de aplicaciones en la Superintendencia de Sociedades son los siguientes:

Adicionalmente, para todas las librerías, dependencias y frameworks que se integren a los proyectos —independientemente del lenguaje base—, cualquier componente nuevo que no esté explícitamente relacionado en este documento deberá ser validado previamente por el equipo de arquitectura, el cual evaluará su seguridad, rendimiento, mantenibilidad y compatibilidad con el ecosistema institucional. Asimismo, estos componentes externos solo podrán utilizar licencias Apache 2.0 o MIT. Queda estrictamente prohibido el uso de librerías con licencias GPL, AGPL, LGPL o cualquier otra que imponga restricciones de copyleft fuerte que puedan comprometer la distribución o el uso de la aplicación.

4.1.1. JAVA:

Al ser un **lenguaje de programación de alto nivel**, robusto, seguro orientado a objetos, ampliamente utilizado para el desarrollo de aplicaciones de todo tipo, desde aplicaciones móviles hasta sistemas empresariales en la superintendencia se pretende utilizar en las Últimas Versiones LTS de licencia libre y usar exclusivamente los siguientes frameworks y/o librerías:

- **SPRING BOOT WEBFLUX**: Es un módulo de Spring Framework diseñado para crear aplicaciones web reactivas y basadas en asincronía. Es parte de Spring Boot, pero se enfoca en aplicaciones que manejan una gran cantidad de solicitudes simultáneas de manera eficiente, utilizando el enfoque reactivo, motivo por el cual se utilizara Para la generación de microservicios o artefactos standalone.

- **SPRING DATA**: Es un proyecto de Spring Framework que facilita el acceso y la manipulación de datos en aplicaciones Java. Su objetivo principal es simplificar el uso de bases de datos en las aplicaciones, independientemente del tipo de base de datos que se utilice (relacional, no relacional, etc.). Spring Data proporciona una serie de tecnologías y herramientas que permiten interactuar con las bases de datos de una manera más eficiente y abstracta, reduciendo la cantidad de código necesario para interactuar con el

almacenamiento de datos. (comunicaciones con el motor de la base de datos.)

- **LOMBOK**: Es una librería de Java que se utiliza para reducir el boilerplate code (código repetitivo o redundante) y mejorar la legibilidad del código, generando automáticamente ciertas clases y métodos comunes, como getters, setters, constructores, toString(), equals(), hashCode(), y otros, a través de anotaciones y en la fábrica se utilizara para la abstracción y polimorfismo.

- **LOG4J2**: Para la generación de archivos logs ya que al ser una biblioteca de registro (logging) para aplicaciones Java permite generar y gestionar logs (registros) de eventos, errores y otros mensajes dentro de la aplicación. **Log4j2** es la versión más reciente del proyecto **Log4j**, que originalmente fue creado por el **Apache Software Foundation**. **Log4j2** mejora la versión anterior, **Log4j**, en varios aspectos, incluyendo el rendimiento, la flexibilidad y la configuración.

- **FREEMAKER**: Para realizar plantillas en html y transformarlas a pdf ya que es una **plantilla motor** (template engine) de Java que se utiliza para generar texto basado en plantillas. Se usa comúnmente en aplicaciones Java para separar la lógica de negocio de la presentación. Con **Freemarker**, puedes crear plantillas de texto que contienen marcadores de posición que serán reemplazados por datos en tiempo de ejecución. Es muy popular en el desarrollo web, especialmente para generar páginas HTML dinámicas, correos electrónicos, o cualquier tipo de archivo de texto basado en datos.

- **APACHE POI**: Es una es una librería de código abierto de **Apache Software Foundation** diseñada para trabajar con documentos de Microsoft Office (como **Excel**, **Word**, y **PowerPoint**) desde aplicaciones Java. POI (Project Object Model) permite la manipulación y creación de archivos de formatos como **.xls, .xlsx, .doc, .docx, .ppt, .pptx**, entre otros, sin necesidad de tener instalado Microsoft Office en el sistema.

- **Apache PDFBox**: Es una herramienta de código abierto mantenida por la Apache Software Foundation, que permite crear, extraer contenido, rellenar formularios, firmar digitalmente y manipular metadatos de documentos PDF, todo bajo la licencia Apache 2.0, la cual es plenamente compatible con la política institucional.

Regla obligatoria de metadatos (JAVA)

Independientemente de la librería utilizada, todo documento PDF generado o modificado por las aplicaciones institucionales **debe** incluir en sus metadatos la identificación del sistema productor. Esta medida permite la trazabilidad y auditoría de los documentos.

Ejemplo en Java (Apache PDFBox):

```
PDDocument document = new PDDocument();
PDDocumentInformation
info = document.getDocumentInformation();
info.setProducer("{nombreAplicacion} - Superintendencia de Sociedades");
```

4.1.2. .NET:

Es un **marco de desarrollo de software** (framework) creado por **Microsoft** que proporciona un entorno de programación para construir aplicaciones de todo tipo, desde aplicaciones de escritorio hasta aplicaciones web, servicios y aplicaciones móviles, en la superintendencia se pretende utilizar en las Últimas Versiones LTS de licencia libre y usar exclusivamente los siguientes frameworks y/o librerías:

- **Net Core**: Al ser un marco de desarrollo multiplataforma, de código abierto y modular, diseñado para ser utilizado en la creación de aplicaciones modernas, especialmente aplicaciones web, de consola, microservicios, APIs y aplicaciones móviles se utilizará para la generación de microservicios o artefactos standalone.

- **Entity framework**: Es un **Object-Relational Mapper (ORM)** para .NET, que facilita la interacción entre las aplicaciones **.NET** y las bases de datos relacionales. EF permite a los desarrolladores trabajar con bases de datos utilizando objetos y clases de **C#** o cualquier otro lenguaje compatible de **.NET**, en lugar de escribir consultas SQL manualmente. Para las comunicaciones con el motor de la base de datos.)

- **Nlog**: Es una **biblioteca de logging** (registro de eventos) para aplicaciones .NET. Su principal objetivo es proporcionar una manera flexible y eficiente de registrar mensajes de log, como información, advertencias, errores y excepciones, durante la ejecución de una aplicación. Esto es útil para la depuración, monitoreo y análisis del comportamiento de las aplicaciones. ara la generación de archivos logs.

- **Razor**: Es un motor de plantillas para aplicaciones web, utilizado principalmente en el desarrollo de aplicaciones web con **ASP.NET**. Permite a los desarrolladores combinar **HTML** y **C#** de manera eficiente para crear páginas dinámicas y generar contenido HTML de forma programática y en la superintendencia se utilizará para realizar plantillas en html y transformarlas a pdf.

- **NPOI**: Es una biblioteca de código abierto en **.NET** que permite leer, escribir y manipular archivos de Microsoft Office (como **Excel, Word** y **PowerPoint**) sin necesidad de tener Microsoft Office instalado en el sistema y en la entidad se utilizara para manipulación de archivos doc, docx, xls, xlsx.

- **QuestPDF y PdfPig**: Se recomienda el uso de QuestPDF (licencia MIT) para la generación de documentos, o PdfPig (licencia Apache 2.0) para la extracción y manipulación de datos. Cualquier otra librería PDF que se pretenda usar en .NET deberá pasar por el proceso de validación del equipo de arquitectura, demostrando el cumplimiento de la licencia permitida.

Regla obligatoria de metadatos (NET):

Independientemente de la librería utilizada, todo documento PDF generado o modificado por las aplicaciones institucionales debe incluir en sus metadatos la identificación del sistema productor. Esta medida permite la trazabilidad y auditoría de los documentos.

Ejemplo en .NET (QuestPDF):

```
.Metadata(meta => meta.Producer("{nombreAplicacion} - Superintendencia de Sociedades"))
Ejemplo en .NET (PdfPig - al modificar metadatos existentes):
using (var document = PdfDocument.Open(file, new ParsingOptions()))
    {
   document.SetProducer("{nombreAplicacion} - Superintendencia de Sociedades");
}
```

4.1.3. Frontend

Es la parte de una aplicación web o móvil que interactúa directamente con los usuarios. Es todo lo que el usuario ve y con lo que interactúa en su navegador o dispositivo, y se encarga de la presentación y la experiencia de usuario (UX). En términos simples, el **frontend** se refiere a la **interfaz de usuario (UI)** y la **lógica del lado del cliente** y se utilizaran las siguientes bibliotecas:

- **React**: permite crear componentes reutilizables que gestionan su propio estado, y luego componer estos componentes para construir interfaces completas. Es conocido por su enfoque basado en **componentes**, lo que significa que una aplicación en React se puede dividir en pequeñas piezas de código independientes y reutilizables y en la entidad se utilizara con estilos de gov.co.

- **Angular**: Es un **framework de desarrollo** de código abierto para crear aplicaciones web, especialmente aplicaciones de una sola página (SPA, por sus siglas en inglés), proporciona una solución integral para desarrollar aplicaciones web, con herramientas para manejar todo desde la interfaz de usuario (UI) hasta la lógica del negocio, el enrutamiento y la gestión de datos y será utilizado Con estilos de gov.co.

Configuración de Aplicaciones Frontend (Angular y React)

Objetivo

Garantizar que las aplicaciones frontend desarrolladas en Angular o React mantengan una configuración estandarizada, segura y parametrizable para los diferentes ambientes definidos por la Entidad.

Separación por Ambientes

Toda aplicación frontend deberá contar con configuraciones independientes para cada ambiente:

Desarrollo (DEV)

Pruebas (QA)

Producción (PRD)

No se permite modificar manualmente archivos de configuración durante el proceso de despliegue. La selección de la configuración deberá realizarse mediante el proceso de construcción (build) o mediante variables de entorno.

Angular

Para aplicaciones Angular se deberá utilizar el mecanismo de archivos de entorno provisto por el framework.

Ejemplo de estructura:

src/environments/
environment.ts
environment.dev.ts
environment.qa.ts
environment.prod.ts
Ejemplo de configuración:

```
export const environment = {
production: true,
keycloak: {
url: '<URL_SERVIDOR_IDENTIDAD>',
realm: '<REALM>',
clientId: '<CLIENT_ID>'
},
sistema: {
tiempoInactividad: <MINUTOS>,
refrescarTokenAntes: <SEGUNDOS>,
refrescarTokenIntervalo: <MILISEGUNDOS>,
intervaloVerificacion: <MILISEGUNDOS>,
tiempoEspera: <MILISEGUNDOS>,
urlAplicacion: '<URL_APLICACION>'
}
};
```

React

Para aplicaciones React se deberán utilizar variables de entorno o archivos de configuración externos que permitan diferenciar los ambientes.

Ejemplo:
.env.development
.env.qa
.env.production
Variables sugeridas:

```
REACT_APP_KEYCLOAK_URL=
REACT_APP_KEYCLOAK_REALM=
REACT_APP_KEYCLOAK_CLIENT_ID=
REACT_APP_KEYCLOAK_TIMEOUT_MS=
REACT_APP_KEYCLOAK_LOGIN_IFRAME_INTERVAL_MS=
REACT_APP_SISTEMA_TIEMPO_INACTIVIDAD_MIN=
```

REACT_APP_SISTEMA_REFRESCAR_TOKEN_ANTES_SEG=
REACT_APP_SISTEMA_REFRESCAR_TOKEN_INTERVALO_MS=
REACT_APP_SUCURSAL_DIGITAL_URL=

Parámetro	Descripción
URL del servidor de identidad	Dirección del proveedor de autenticación.
Realm	Dominio de autenticación asignado por la Entidad.
Client ID	Identificador único de la aplicación.
Tiempos de sesión	Parámetros de control de expiración e inactividad.
Configuración de renovación de token	Intervalos y umbrales de actualización de credenciales.

Restricciones de Seguridad

- No se deberán almacenar contraseñas, secretos, certificados o credenciales en el código fuente.
- Toda configuración deberá encontrarse parametrizada y separada por ambiente.
- Los valores específicos de cada ambiente serán suministrados por la Entidad.
- Las URL institucionales deberán utilizar los dominios oficiales proporcionados por la Entidad.
- Para ambientes expuestos a Internet se deberá utilizar exclusivamente protocolo HTTPS.
- Los dominios institucionales publicados deberán corresponder al dominio gubernamental.gov.co suministrado y administrado por la Entidad.
- Ningún desarrollador deberá modificar manualmente configuraciones de producción sin seguir el procedimiento formal de gestión de cambios.

- Los artefactos generados para producción no deberán contener referencias a ambientes de desarrollo o pruebas.

Buenas Prácticas

1. Centralizar toda la configuración en archivos de entorno.
2. Evitar valores hardcodeados dentro de componentes o servicios.
3. Mantener nombres de variables homogéneos entre Angular y React.
4. Documentar cualquier nuevo parámetro de configuración incorporado al proyecto.
5. Validar durante el arranque de la aplicación que todos los parámetros obligatorios hayan sido configurados.
6. Utilizar mecanismos de CI/CD para inyectar o seleccionar la configuración correspondiente a cada ambiente.

5. Convenios de Nomenclatura

5.1. JAVA

Nombres de Clases:

- Utilizar **PascalCase**, comenzando con una letra **mayúscula**.
- Los nombres deben ser sustantivos o frases nominales, que describan claramente qué representa la clase.

Nombres de Métodos:

- Utilizar **CamelCase**, comenzando con una letra **minúscula**.
- Los nombres de métodos deben ser verbos o frases verbales, describiendo lo que hace el método.

Nombres de Variables:

- Utilizar CamelCase, comenzando con una letra **minúscula**.

- Los nombres de las variables deben ser descriptivos y reflejar claramente su propósito.

Nombres de Constantes:

- Utilizar todas las letras en **mayúsculas** con palabras separadas por un **guion bajo** (_).
- Generalmente se declaran como static final y son valores que no cambian.

Nombres de Paquetes:

- Utilizar solo **letras minúsculas** y **palabras separadas por puntos** (.).
- Generalmente el nombre de un paquete comienza con el dominio invertido de la empresa u organización seguido del nombre del proyecto o módulo.

Nombres de Interfaces:

- Utilizar **PascalCase**, comenzando con una letra **mayúscula**.
- El nombre de la interfaz debe ser un adjetivo o describir el comportamiento que implementan las clases que la usen.

Nombres de Enumeraciones (enums):

- El nombre de la enumeración sigue las mismas reglas que las clases (**PascalCase**, **mayúscula inicial**).
- Los valores dentro de la enumeración se escriben en mayúsculas separadas por guion bajo si es necesario.

Nombres de Parámetros:

- Se utiliza **CamelCase**, comenzando con una letra **minúscula**.
- Los nombres de parámetros deben ser descriptivos y reflejar el valor que representan.

Nombres de Tipos Genéricos:

- Se suele usar una sola letra mayúscula para representar tipos genéricos.

Elemento	Convención	Ejemplo Correcto

Clases	PascalCase	OrderProcessor
Interfaces	PascalCase	IRepository
Métodos	camelCase	calculateTotal()
Variables	camelCase	totalAmount
Constantes	UPPER_SNAKE_CASE	MAX_RETRIES
Paquetes (packages)	lowercase	com.company.project.module
Enumeraciones	PascalCase (tipo) + UPPER_SNAKE_CASE (valores)	OrderStatus PENDING, COMPLETED
Parámetros	camelCase	customerId
Campos Privados	camelCase (opcional prefijo _)	_totalAmount o totalAmount
Propiedades (Getters/Setters)	camelCase para métodos con prefijo get/set	getName(), setName()
Genéricos	Single uppercase letter o descriptivos	T, E, K, V

5.2. C#:

Nombres de Clases:

- Convención: **PascalCase**.
- Los nombres deben ser sustantivos o frases que describan la funcionalidad o entidad.

Nombres de Métodos:

- Convención: **PascalCase**.
- Los métodos deben ser verbos o frases que describan una acción.
- Deben transmitir claramente lo que hace el método.

Nombres de Variables:

- Convención: **camelCase**.
- Los nombres de variables deben ser descriptivos y reflejar su propósito.
- Las variables no deben incluir prefijos relacionados con su tipo de dato (no usar notación húngara).

Variables locales

- Usan **camelCase**.
- Deben ser breves, pero suficientemente descriptivas.

Variables privadas (campos)

- Usan **camelCase** con un prefijo _ (subrayado).
- Esto ayuda a distinguirlas de otras variables y propiedades.

Variables estáticas

- Usan **camelCase** con prefijo _ para mantener consistencia con los campos privados.
- Si son constantes, usan **UPPER_SNAKE_CASE**.

Nombres de Constantes:

- Utilizar todas las letras en **mayúsculas** con palabras separadas por un **guion bajo** (_).
- Convención: UPPER_SNAKE_CASE.
- Las constantes usan const o readonly y suelen representar valores inmutables.

Nombres de Paquetes (Namespaces):

- Convención: **PascalCase**.
- Siguen la estructura jerárquica con nombres descriptivos separados por puntos.
- Utilizan un esquema que refleja la organización del proyecto.

Nombres de Interfaces:

- Convención: **PascalCase** con el prefijo I.
- El nombre debe describir el contrato que define la interfaz.

Nombres de Enumeraciones (enums):

- Convención: **PascalCase** para el nombre de la enumeración.
- Los valores de la enumeración también usan PascalCase.
- Los nombres deben ser sustantivos o describir un conjunto de valores relacionados.

Nombres de Parámetros:

- Convención: camelCase.
- Los parámetros deben ser descriptivos y reflejar el propósito del valor que representan.

Nombres de Tipos Genéricos:

- Se suele usar una sola letra mayúscula para representar tipos genéricos.

Elemento	Convención	Ejemplo
Clases	PascalCase	OrderProcessor
Métodos	PascalCase	GetCustomer()
Constantes	UPPER_SNAKE_CASE	MAX_RETRIES
Paquetes/Namespaces	PascalCase	MyCompany.Project.Module
Interfaces	PascalCase con I	IOrderProcessor
Enumeraciones	PascalCase	OrderStatus
Variables Locales	camelCase	totalAmount
Campos Privados	camelCase con _	_customerName
Propiedades	PascalCase	TotalAmount
Parámetros	camelCase	orderId

5.3. Servicios Web (Rest)

Con el propósito de fortalecer la gobernanza tecnológica de la superintendencia de sociedades, la mantenibilidad y la estandarización de los servicios expuestos en la arquitectura de microservicios de la organización, se define un estándar corporativo para el nombramiento de las URLs de las APIs REST.

Este estándar tiene como objetivo establecer una convención única, clara y consistente para la construcción de rutas de acceso a los servicios, facilitando:

- La comprensión y reutilización de las APIs por parte de los equipos de desarrollo.
- La integración entre aplicaciones internas y externas.
- La administración centralizada mediante API Gateway.
- La trazabilidad y documentación técnica de los servicios.
- La reducción de inconsistencias entre proyectos y equipos.
- La aplicación homogénea de políticas de seguridad y monitoreo.

La convención definida se adapta a la arquitectura actual de la organización, la cual se basa en microservicios internos publicados a través de un API Gateway corporativo, y contempla las directrices de seguridad establecidas por la organización, incluyendo el uso exclusivo de los métodos **HTTP GET** y **POST**.

Adicionalmente, este estándar busca evitar prácticas que generen URLs extensas, ambiguas o dependientes de lógica de negocio específica, promoviendo rutas orientadas a recursos funcionales y acciones controladas, bajo criterios de simplicidad, legibilidad y escalabilidad.

La adopción de este estándar permitirá que todos los servicios expuestos mantengan una estructura uniforme, facilitando futuras migraciones, procesos de modernización tecnológica, automatización de documentación **OpenAPI/Swagger** y controles de calidad sobre las APIs institucionales.

5.3.1. Estructura base

Interna

/micro-{**aplicacion**}/{recurso}/{accion}

Ejemplo
/micro-**sucursal-digital**/aplicaciones/consultar

Externa (Gateway)

/api/v2/{**sucursal-digital**}/{accion}
Ejemplo
/api/v2/**sucursal-digital**/aplicaciones/consultar

5.3.2. Convención de nombres

Recursos
minúsculas
kebab-case
sustantivos
sin camelCase

Ejemplos:
accion-modulo
aplicaciones
personas
radicados
tipos-identificacion

5.3.3. Acciones permitidas

Como usan GET/POST únicamente, entonces sí deben existir acciones explícitas. Eso cambia el estándar.

5.3.4. Estándar CRUD corporativo

Consultar
Listar
GET /aplicaciones/consultar
Obtener por ID

GET /aplicaciones/consultar/{id}

Filtro simple
Opción 1
GET /aplicaciones/consultar-por-codigo
Opción 2
GET /aplicaciones/consultar?codigo=ABC

Crear
POST /aplicaciones/crear

Actualizar
POST /aplicaciones/actualizar/{id}

Eliminar lógico
POST /aplicaciones/eliminar/{id}

Operaciones especiales
POST /radicados/aprobar/{id}
POST /radicados/rechazar/{id}
POST /usuarios/desbloquear/{id}
POST /usuarios/reestablecer-clave/{id}

Búsquedas complejas
Cuando hay caracteres especiales, payload grande o filtros complejos usar:
POST /radicados/consultar
Body:

```
{
  "numeroRadicado": "...",
  "fechaInicio": "...",
  "estado": "..."
}
```

Nota: Se debe evitar verbos redundantes para evitar mezcla de lógica o url enormes que sea difícil la gobernanza.

Antes
/obtenerAplicacionPorCodigo
/consultarTramitesPorRadicadoPadre
Con el estándar
GET /aplicaciones/consultar-por-codigo
GET /tramites/consultar-por-radicado-padre
GET /tramites/consultar?radicadoPadre=123

Regla	Valor
Métodos HTTP	GET y POST
URLs	minúsculas
Separador	kebab-case
Recursos	sustantivos
Versionado	API Gateway

Acciones estándar

Acción	Método	URL
Consultar lista	GET	/usuarios/consultar
Consultar por id	GET	/usuarios/consultar/{id}
Crear	POST	/usuarios/crear
Actualizar	POST	/usuarios/actualizar/{id}
Eliminar	POST	/usuarios/eliminar/{id}

Búsquedas

Operación	Método	URL
Radicar entrada	POST	/radicaciones
Consultar	GET	/radicaciones/consultar

Ejemplo sucursal digital

Actual
/micro-sucursal-digital/aplicacion/obtenerAplicaciones
/micro-sucursal-digital/aplicacion/crearAplication
/micro-sucursal-digital/aplicacion/actualizarAplication/{id}

Estandar
/micro-sucursal-digital/aplicaciones/consultar
/micro-sucursal-digital/aplicaciones/consultar/{id}
/micro-sucursal-digital/aplicaciones/crear
/micro-sucursal-digital/aplicaciones/actualizar/{id}
/micro-sucursal-digital/aplicaciones/eliminar/{id}

Estas acciones que se muestra a continuación describen los escenarios en las cuales se pueden aplicar adicionalmente a ello, se pueden utilizar en los scopes del resource serve en la capa oauth de cada microservicio.

Acción	Descripción	Casos de uso
consultar	Obtener información sin modificar estado	búsquedas, consultas
crear	Registrar un nuevo recurso	creación de entidades
actualizar	Modificar información existente	edición de datos
eliminar	Remover lógica o físicamente un recurso	borrado
aprobar	Autorizar una operación o flujo	aprobación de trámites

rechazar	Negar o devolver una solicitud	rechazo de procesos
validar	Ejecutar reglas o verificaciones	validaciones de negocio
generar	Construir un recurso derivado	generación de reportes, generar radicados
descargar	Obtener un archivo existente	descarga PDF/Excel
exportar	Extraer información hacia formato externo	Excel, CSV
importar	Cargar información desde fuente externa	cargas masivas
cargar	Subir archivos o estructuras	uploads
reprocesar	Ejecutar nuevamente un proceso	reintentos
sincronizar	Actualizar información entre sistemas	integraciones
finalizar	Cerrar un proceso o flujo	cierre de casos
anular	Invalidar un proceso previamente ejecutado	anulaciones

5.3.5. Estándar de Respuesta para Servicios

Objetivo

Garantizar que todos los servicios REST de la organización expongan respuestas homogéneas y permitan a las aplicaciones consumidoras identificar fácilmente si una operación fue exitosa, presentó errores de validación o falló por causas técnicas.

Estructura General de Respuesta

Todos los servicios deberán retornar una estructura estándar compuesta por:

- **codigo**: Código de resultado de la operación.
- **mensaje**: Descripción funcional del resultado.

- **objeto**: Información retornada por el servicio.

Ejemplo:

```
{
  "codigo": 0,
  "mensaje": "La operación se ha realizado exitosamente.",
  "objeto": {}
}
```

Códigos Corporativos de Respuesta

Código	Tipo	Descripción
0	Exitoso	La operación se ejecutó correctamente.
1	Error Técnico	Ocurrió una falla técnica o inesperada.
2	Error de Validación o Negocio	La solicitud no cumple reglas funcionales o de validación.
3	Sin Resultados	La consulta fue exitosa pero no encontró información.

Manejo de Errores Técnicos

Los errores técnicos corresponden a situaciones no controladas por la lógica de negocio, por ejemplo:

- NullPointerException.
- SQLException.
- Timeout de servicios externos.
- Problemas de conectividad.
- Errores de serialización.
- Excepciones no contempladas.

Lineamientos

- No deben ser capturados ni procesados individualmente en cada controlador.
- Deben ser gestionados mediante un mecanismo centralizado de manejo de excepciones (Interceptor, Exception Handler o Controller Advice).
- Deben retornar el código corporativo 1.
- No deben exponer detalles internos de la aplicación al consumidor.
- El detalle técnico debe registrarse únicamente en los logs de la plataforma.

Ejemplo:

```
{
  "codigo": 1,
  "mensaje": "Ha ocurrido un error, por favor intentar más tarde."
}
```

Manejo de Validaciones y Reglas de Negocio

Las validaciones funcionales corresponden a errores esperados dentro del proceso de negocio.
Ejemplos:

- El radicado no existe.
- El usuario no tiene permisos.
- La sociedad no está habilitada.
- El archivo tiene una extensión no permitida.
- Ya existe un documento principal para la solicitud.

Lineamientos

- Deben ser controladas explícitamente por la capa de negocio.
- No deben generar excepciones técnicas genéricas.
- Deben retornar el código corporativo 2.
- Deben retornar mensajes funcionales claros para el usuario.

Ejemplos:

```
{
  "codigo": 2,
```

"mensaje": "El radicado no existe en el gestor documental."
}
{
"codigo": 2,
"mensaje": "La extensión del archivo a radicar no está permitida."
}
{
"codigo": 2,
"mensaje": "La sociedad no se encuentra habilitada para realizar la radicación."
}

Consultas sin Resultados

Cuando una consulta sea ejecutada correctamente pero no existan registros asociados a los criterios enviados, deberá retornarse el código corporativo 3.
Ejemplo:

{
"codigo": 3,
"mensaje": "Ha ocurrido un error, la consulta no ha arrojado resultados."
}

Este escenario no deberá considerarse un error técnico.

Registro y Trazabilidad

Toda excepción técnica deberá registrarse en la plataforma de monitoreo y observabilidad institucional (tablero de logs), incluyendo como mínimo:

- Fecha y hora.
- Servicio.
- Método invocado.
- Identificador de correlación.
- Mensaje de error.
- Stack Trace.

Nota: Los mensajes enviados al cliente nunca deberán contener información sensible de infraestructura, consultas SQL, rutas del servidor o detalles internos

de implementación.

6. Versionamiento de Aplicaciones

Con el propósito de garantizar la trazabilidad de los cambios, facilitar la identificación de las versiones desplegadas y mantener un control uniforme sobre los componentes de las aplicaciones, todos los proyectos deberán implementar versionamiento semántico (Semantic Versioning).

El versionamiento deberá aplicarse a los componentes **Front-End**, **Back-End** y Base de Datos, de acuerdo con la naturaleza de los cambios realizados. El formato de versión deberá seguir la estructura:

MAJOR.MINOR.PATCH

Por ejemplo:

Versión 2.1.0

6.1. Versionamiento semántico

El número de versión estará compuesto por tres elementos:

Componente	Descripción	Ejemplo
MAJOR	Se incrementa cuando se realizan cambios incompatibles que pueden afectar el funcionamiento de versiones anteriores.	2.0.0
MINOR	Se incrementa cuando se agregan nuevas funcionalidades manteniendo la compatibilidad con la versión anterior.	2.1.0

PATCH	Se incrementa cuando se realizan correcciones de errores, ajustes menores o cambios que no incorporan nuevas funcionalidades.	2.1.1

La versión deberá iniciar, como referencia, en 1.0.0 cuando se libere una primera versión estable del aplicativo.

Criterios para el incremento de versión

Se deberá incrementar el número de versión de acuerdo con el impacto del cambio:

MAJOR – Cambios incompatibles

Se deberá incrementar MAJOR cuando se introduzcan cambios que puedan requerir modificaciones en otros componentes o consumidores de la aplicación.

Ejemplos:

- Modificación de contratos de servicios que rompan la compatibilidad existente.
- Eliminación o modificación incompatible de endpoints.
- Cambios que requieran modificaciones obligatorias en el Front-End.
- Eliminación de funcionalidades existentes.
- Cambios incompatibles en estructuras de datos.

Ejemplo:

1.4.2 → 2.0.0

MINOR – Nuevas funcionalidades compatibles

Se deberá incrementar MINOR cuando se incorporen nuevas funcionalidades o capacidades sin romper la compatibilidad existente.

Ejemplos:

- Incorporación de nuevos módulos.
- Nuevos endpoints compatibles.

- Nuevas funcionalidades para los usuarios.
- Nuevos campos que no afecten el funcionamiento existente.

Ejemplo:

1.4.2 → 1.5.0

PATH – Correcciones y ajustes menores

Se deberá incrementar PATCH cuando se realicen correcciones o ajustes que no modifiquen de manera significativa el comportamiento o contrato de la aplicación.

Ejemplos:

- Corrección de errores.
- Ajustes menores de lógica.
- Correcciones de validaciones.
- Correcciones visuales en el Front-End.
- Ajustes menores de configuración.

Ejemplo:

1.4.2 → 1.4.3

6.2. Versionamiento en Front-End

Las aplicaciones desarrolladas en React y Angular deberán mostrar de manera visible la versión actual del aplicativo dentro de la interfaz de usuario.

La versión deberá utilizar el formato:

Versión MAJOR.MINOR.PATCH

Por ejemplo:

Versión 2.1.0

La ubicación deberá ser consistente y preferiblemente discreta, permitiendo identificar fácilmente la versión del aplicativo durante actividades de soporte, pruebas y revisión técnica.

React

Ejemplo de implementación:

```
<div className="d-flex justify-content-center text-muted">
  <small>Versión 2.1.0</small>
</div>
```

Angular

Ejemplo de implementación:

```
<div class="d-flex justify-content-center text-muted">
  <small class="text-muted">Versión 2.1.0</small>
</div>
```

La versión mostrada deberá corresponder con la versión que se encuentre desplegada en el ambiente correspondiente.

6.2. Versionamiento en Back-End

Los servicios desarrollados en el Back-End deberán exponer la información de versión de la aplicación mediante la documentación Swagger/OpenAPI.

La versión deberá estar disponible de forma visible en la documentación del servicio, permitiendo identificar fácilmente la versión del API que se encuentra desplegada.

La definición de la versión deberá utilizar el esquema:

MAJOR.MINOR.PATCH

Por ejemplo:

2.1.0

Esta información deberá mantenerse actualizada de acuerdo con los cambios realizados en los servicios.

De manera complementaria, cuando el proyecto utilice versionamiento de API mediante URL, rutas o encabezados, este deberá mantenerse alineado con la versión documentada en **Swagger/OpenAPI.**

Ejemplo conceptual:

info:

title: API de Gestión

version: 2.1.0

description: API para la gestión de información del aplicativo.

La versión registrada en Swagger deberá corresponder a la versión real del servicio desplegado, evitando que la documentación presente una versión diferente a la que se encuentra disponible en el ambiente.

6.3. Versionamiento de Base de Datos

Los cambios realizados sobre la Base de Datos deberán aplicar igualmente el esquema de versionamiento semántico, teniendo en cuenta el impacto y la compatibilidad de cada modificación.

MAJOR – Cambios incompatibles

Se deberá incrementar MAJOR cuando se realicen cambios incompatibles con la estructura o funcionamiento existente de la base de datos.

Ejemplos:

- Eliminación de tablas.
- Eliminación de columnas utilizadas por la aplicación.
- Modificación incompatible de tipos de datos.
- Cambios estructurales que requieran modificar obligatoriamente los componentes que consumen la base de datos.

Ejemplo:

1.2.3 → 2.0.0

MINOR – Cambios compatibles

Se deberá incrementar MINOR cuando se agreguen estructuras que mantengan la compatibilidad con la versión existente.

Ejemplos:

Creación de nuevas tablas.

Adición de nuevas columnas que no afecten los datos existentes.

Creación de nuevos índices.

Incorporación de nuevas estructuras necesarias para funcionalidades nuevas.

Ejemplo:

1.2.3 → 1.3.0

PATCH – Correcciones y ajustes menores

Se deberá incrementar PATCH cuando se realicen modificaciones menores que no impliquen cambios incompatibles en la estructura de la base de datos.

Ejemplos:

- Corrección de datos.
- Ajustes menores sobre registros.
- Corrección de scripts.
- Modificaciones que no alteren el contrato estructural existente.

Ejemplo:

1.2.3 → 1.2.4

6.4. Consistencia entre componentes

Cuando una funcionalidad involucre cambios en varios componentes de la solución, el equipo de desarrollo deberá evaluar el impacto del cambio y actualizar las versiones correspondientes.

Como mínimo, deberá existir trazabilidad entre:

Front-End → Back-End → Base de Datos

Por ejemplo, para una nueva funcionalidad que requiera una nueva pantalla en React, un nuevo endpoint en el Back-End y una nueva tabla en la Base de Datos, se deberá identificar el cambio como una nueva funcionalidad compatible y realizar el incremento de versión MINOR en los componentes afectados, de acuerdo con la estrategia de versionamiento definida para el proyecto.

En caso de que el cambio genere incompatibilidad con versiones anteriores, deberá clasificarse como MAJOR, independientemente de que la modificación corresponda a Front-End, Back-End o Base de Datos.

6.5. Metadato description

Las aplicaciones deberán incluir adicionalmente el metadato HTML description, el cual deberá contener una descripción corta y clara de la finalidad principal del aplicativo.

Ejemplo:

```
<meta
  name="description"
  content="Aplicación para la gestión y consulta de calendarios institucionales."
>
```

La descripción deberá ser específica para cada aplicativo y mantenerse actualizada cuando cambie significativamente su propósito o funcionalidad principal.

6.6. Requisito de cumplimiento

Durante las actividades de desarrollo, pruebas, despliegue y revisión técnica se deberá verificar que:

- El Front-End (React/Angular) muestre la versión actual de la aplicación.
- El Back-End registre la versión correspondiente en Swagger/OpenAPI.
- La Base de Datos utilice versionamiento semántico para sus cambios.
- Las versiones sigan el formato **MAJOR.MINOR.PATCH.**
- El incremento de versión corresponda al impacto real del cambio.

La versión documentada corresponda con la versión efectivamente desplegada.

El Front-End incluya el metadato "description" con una descripción corta y representativa de la aplicación.

7. Mensajería

El objetivo del estándar de nombramiento es garantizar orden, trazabilidad, escalabilidad y mantenibilidad dentro de la plataforma de mensajería. Un nombre claro y estructurado permite identificar rápidamente el propósito, responsable, ambiente y tipo de cada cola o exchange. Además, facilita la automatización (DLQ, retry), minimiza errores humanos y permite aplicar políticas de seguridad basadas en prefijos.

7.1. Estructura del nombramiento

Los elementos se nombran usando el siguiente patrón:
<ambiente>.<sistema>.<módulo>.<acción>.<consumidor>

ambiente: dev, test, prod
dominio: iniciativa o macroproyecto (ej. sucursal-digital)
módulo: modulo el cual pertenece la cola(ej: radicación)
acción: acción que realizara la cola. (ej. validación-archivos, procesar-archivos, insertar-documento)
consumidor: microservicio responsable (ej. ms1, ms2)

7.2. Tipos de colas

Cola principal
Recibe los mensajes normales del flujo.

Cola Retry
Recibe mensajes que deben reintentarse después de cierto tiempo.
Se usa con exchanges del tipo delayed o con TTL + dead-lettering.

Cola DLQ (Dead Letter Queue)
Almacena mensajes que no pudieron ser procesados después de los reintentos. Permite auditoría, inspección manual y evitar pérdida de información.

7.3. Tipos de Exchanges

direct: dirigido por routing key exacta
topic: patrones de routing
fanout: para difusión
x-delayed-message: permite reintentos con tiempos configurables (retry)

Ejemplo oficial siguiendo el estándar

Exchange principal
test.mensajeria-archivos.radicacion.direct.procesar-archivos
Cola principal
test.mensajeria-archivos.radicacion.procesar-archivos.ms2
Cola Retry
test.mensajeria-archivos.radicacion.procesar-archivos.ms2.retry
Cola DLQ
test.mensajeria-archivos.radicacion.procesar-archivos.ms2.dlq
Routing key
procesar-archivos.ms2

8. Lineamiento para la documentación del proyecto – README.md

Todos los proyectos deberán contar en la raíz del repositorio con un archivo README.md, el cual deberá contener la información necesaria para comprender el propósito de la aplicación, sus características principales, los requisitos para su ejecución, el procedimiento para obtener el código fuente, las instrucciones de despliegue y las referencias a la documentación técnica.

El archivo README.md deberá mantenerse actualizado con los cambios realizados en el proyecto y deberá utilizar, como mínimo, la siguiente

estructura.

8.1. Introducción

Deberá contener una descripción breve del proyecto, indicando su propósito principal y el problema o necesidad que busca solucionar.

Ejemplo:

Este proyecto se crea para realizar el procesamiento de los diferentes archivos para su envío al SGDEA.

8.2. Características del proyecto

Deberá describir las principales funcionalidades o responsabilidades de la aplicación.

Ejemplo:

El objetivo del proyecto es:

- Consultar el estado de la radicación para determinar si el archivo puede ser enviado.
- Enviar el archivo al SGDEA.
- Realizar reintentos en caso de presentarse un error durante el envío.
- Registrar las peticiones que presenten errores en la cola correspondiente, incluyendo el mensaje de error generado.

8.3. Tecnologías y versiones

Se deberán especificar las principales tecnologías utilizadas en el proyecto y sus respectivas versiones.

Ejemplo:

Java: 21

Spring Boot: 3.4

Maven: 3.8.4

WildFly: 35.0.1

Las versiones registradas en el README.md deberán mantenerse alineadas con las versiones realmente utilizadas por el proyecto.

8.4. Prerrequisitos

Se deberán relacionar las herramientas, plataformas, dependencias o configuraciones necesarias para ejecutar, compilar y desplegar el proyecto.

Ejemplo:

Java 21

Maven 3.4

WildFly 35.0.0

Acceso a los repositorios o servicios requeridos por la aplicación.

Configuración de las variables de entorno necesarias.

Cuando aplique, también deberán documentarse configuraciones adicionales, como variables de entorno, archivos de configuración, certificados o credenciales requeridas para la ejecución.

8.5. Obtener el código fuente

Deberá indicarse el procedimiento para obtener el código fuente desde el repositorio institucional.

Ejemplo:

git clone <URL_DEL_REPOSITORIO>

Posteriormente, ingresar al directorio del proyecto:

cd integracion-gedess

La URL del repositorio deberá corresponder al repositorio oficial del proyecto.

8.6. Compilación del proyecto

Se deberán documentar los comandos necesarios para compilar o construir la aplicación.

Ejemplo para proyectos Maven:

mvn clean install -DskipTests=true

Si el proyecto requiere comandos adicionales, estos deberán ser incluidos en esta sección.

8.7. Despliegue

Se deberá documentar el procedimiento necesario para realizar el despliegue de

la aplicación.

Para aplicaciones que generen un archivo WAR, se deberá indicar la ubicación del archivo generado y el servidor de aplicaciones utilizado.

Ejemplo:

Una vez ejecutado el proceso de compilación, se generará el archivo .war dentro de la carpeta:

target/

El archivo .war deberá ser desplegado en la carpeta deployments de WildFly:

wildfly-35.0.1.Final_back\standalone\deployments

8.8. Ejecución de la aplicación

Se deberá documentar el procedimiento para iniciar la aplicación y las herramientas requeridas.

Ejemplo para WildFly:

wildfly-35.0.1.Final_back\bin\standalone.bat

Cuando existan configuraciones adicionales necesarias para iniciar correctamente la aplicación, estas deberán documentarse en esta sección.

8.9. Documentación de servicios – Swagger/OpenAPI

Cuando el proyecto exponga servicios REST, deberá incluirse la ubicación de la documentación de Swagger/OpenAPI.

Se deberá indicar la URL correspondiente al ambiente local y, cuando aplique, las URL de los diferentes ambientes.

Ejemplo:

Swagger:

http://localhost:8080/api/v1/swagger-ui/index.html

La versión registrada en Swagger/OpenAPI deberá corresponder con el versionamiento definido para el Back-End.

8.10. URL de los ambientes

El README.md deberá incluir las URL correspondientes a los ambientes en los que se encuentre desplegada la aplicación.

Ambiente	URL
Desarrollo	URL del ambiente de desarrollo
Pruebas	URL del ambiente de pruebas
Producción	URL del ambiente de producción

Cuando un ambiente no aplique para el proyecto, deberá indicarse explícitamente.

8.11. Configuración adicional

Cuando el proyecto requiera configuraciones específicas para su ejecución o despliegue, estas deberán documentarse.

Por ejemplo:

- Variables de entorno.
- Parámetros de configuración.
- Conexiones a bases de datos.
- Servicios externos.
- Colas o sistemas de mensajería.
- Certificados.
- Rutas de archivos.
- Configuraciones específicas por ambiente.
- No se deberán almacenar en el README.md contraseñas, tokens, llaves privadas, secretos ni información sensible.

8.12. Estructura mínima obligatoria del README.md

Como mínimo, todos los proyectos deberán contar con las siguientes secciones:

```
# Nombre del proyecto
## Introducción
## Características del proyecto
## Tecnologías y versiones
## Prerrequisitos
## Obtener el código fuente
## Compilación del proyecto
## Despliegue
```

Ejecución de la aplicación
Documentación de servicios – Swagger/OpenAPI
URL de los ambientes
Configuración adicional

La estructura podrá ampliarse de acuerdo con las necesidades particulares del proyecto. Por ejemplo, podrán incorporarse secciones de arquitectura, estructura de directorios, pruebas, solución de problemas, dependencias externas, monitoreo o contactos técnicos.

8.13. Criterios de cumplimiento

Durante las revisiones técnicas se deberá verificar que:

El repositorio contenga un archivo README.md en su raíz.

La documentación describa claramente el propósito y las características del proyecto.

- Se encuentren documentadas las tecnologías y versiones utilizadas.
- Se indiquen los prerrequisitos para ejecutar el proyecto.
- Se documente el procedimiento para obtener el código fuente.
- Se documenten los comandos necesarios para compilar y ejecutar la aplicación.
- Se documente el procedimiento de despliegue.
- Se incluya la URL de Swagger/OpenAPI cuando el proyecto exponga servicios.
- Se registren las URL de los ambientes de Desarrollo, Pruebas y Producción, cuando apliquen.
- La información del README.md se encuentre actualizada y corresponda con la configuración real del proyecto.
- No se incluyan credenciales, contraseñas, tokens o secretos dentro del archivo.

9. Lineamientos de Registro de Logs para Aplicaciones y Microservicios

9.1. Objetivo

Establecer los lineamientos para la generación, almacenamiento, rotación, protección y centralización de logs en las aplicaciones y microservicios de la organización, garantizando trazabilidad, auditoría, soporte operativo y monitoreo centralizado.

9.2. Configuración General

Ruta de almacenamiento

La ubicación de los archivos de log debe ser configurable mediante variables de entorno o archivos de configuración propios de cada aplicación.

No se deben utilizar rutas codificadas dentro del código fuente.

Nombre del archivo

El archivo principal de log deberá tener como nombre el identificador de la aplicación o microservicio.

Ejemplos:

calendarioEventos.log

correoMasivo.log

certificadosEnLinea.log

conciliacion.log

Rotación y compresión

Las aplicaciones deberán implementar rotación automática de archivos de log.

Los archivos históricos deberán almacenarse comprimidos en formato GZIP (.gz) o equivalente.

Formato recomendado:

<nombre-aplicacion>-YYYY-MM-DD-N.log.gz

Ejemplo:

calendarioEventos-2026-06-04-1.log.gz

calendarioEventos-2026-06-04-2.log.gz

Retención

Se recomienda conservar los archivos de log durante un periodo mínimo de 30 días.

La eliminación automática de archivos antiguos deberá ser configurable según las necesidades operativas de cada sistema.

Configuración por Ambiente

Las aplicaciones deberán soportar configuraciones independientes para cada ambiente:

Desarrollo (DEV)

Pruebas (QA)

Producción (PROD)

Las diferencias podrán incluir:

- Nivel de logging.
- Destino de almacenamiento.
- Servidores de centralización de logs.
- Políticas de retención.

Protección de Información Sensible

Está prohibido registrar en logs información sensible o datos personales.

No deberán registrarse:

- Credenciales
- Contraseñas.

- Tokens de autenticación.
- Claves API.
- Secretos de aplicación.
- Información Personal
- Nombres completos.
- Correos electrónicos.
- Direcciones físicas.
- Números telefónicos.
- Números de identificación.
- Información Financiera
- Números de cuentas bancarias.
- Datos de tarjetas de crédito o débito.
- Información de pagos.
- Información Técnica Sensible
- Archivos codificados en Base64.
- Documentos completos.
- Contenido binario.
- Certificados digitales.
- Llaves privadas.

Cuando sea necesario registrar información para trazabilidad, deberá aplicarse enmascaramiento u ofuscación de los datos.

Ejemplos:

usuario: y*****a

correo: y******@supersociedades.gov.co

cuenta: ******7890

Formato Estándar de Logs

Todos los errores registrados deberán seguir una estructura uniforme.

Campos obligatorios:

Campo	Descripción
FECHA	Fecha y hora del evento
NIVEL	TRACE, DEBUG, INFO, WARN, ERROR, FATAL
SERVIDOR	Nombre o IP del servidor

APLICACION	Nombre de la aplicación o microservicio
USUARIO	Usuario asociado a la operación (si aplica)
IP_USUARIO	Dirección IP del usuario
CLASE	Clase donde ocurrió el evento
METODO	Método ejecutado
LINEA	Línea aproximada donde ocurrió el error
ID	Identificador único de trazabilidad (UUID)
MENSAJE	Mensaje general del error
MENSAJE_ESPECIFICO	Descripción detallada de la excepción
CRITICIDAD	ALTA, MEDIA o BAJA
PETICION	Parámetros de entrada depurados y sin información sensible

Ejemplo

2026-06-04 06:18:21.523 ERROR SSWV2-SDI03 :
SERVIDOR: 192.168.195.215,
APLICACION: siis,
USUARIO: usuarioSiis,
IP_USUARIO: 127.0.0.1,
CLASE: GedessServiceImpl,
METODO: consultarPorNumeroRadicacion,
LINEA: 0,
ID: ddec4584-581f-41cc-8253-db7e1c0a7ff0,
MENSAJE: No se encontró la radicación,
MENSAJE_ESPECIFICO: No se encontró la radicación,
CRITICIDAD: ALTA,
PETICION: Metodo Rest POST - numeroRadicacion=2025-01-254706

Niveles de Logging

Nivel configurado	Muestra	Descripción

TRACE	TRACE, DEBUG, INFO, WARN, ERROR, FATAL	Información detallada para diagnóstico profundo.
DEBUG	DEBUG, INFO, WARN, ERROR, FATAL	Información utilizada durante desarrollo y depuración.
INFO	INFO, WARN, ERROR, FATAL	Eventos normales de operación.
WARN	WARN, ERROR, FATAL	Situaciones anómalas que no afectan la continuidad del servicio.
ERROR	ERROR, FATAL	Errores que afectan una operación específica.
FATAL	FATAL	Errores críticos que comprometen la estabilidad o disponibilidad del sistema.

Centralización de Logs

Todos los microservicios y aplicaciones deberán enviar sus eventos al tablero corporativo de monitoreo basado en OpenSearch/ELK. Metadatos obligatorios.

Cada evento enviado deberá incluir como mínimo:

Campo	Descripción
service	Nombre del microservicio o aplicación
timestamp	Fecha y hora del evento
environment	Ambiente (dev, qa, prod)
hostname	Nombre del servidor origen

Ejemplo

```
{
  "service": "tratamiento-datos",
  "timestamp": "2026-06-04T06:18:21.523-0500",
  "environment": "prod",
  "hostname": "SSWV2-SDI03"
}
```

El valor de service deberá corresponder exactamente al nombre del

microservicio o aplicación registrado en la arquitectura institucional.

Buenas Prácticas

- Utilizar logging asíncrono cuando sea posible.
- Registrar únicamente información útil para soporte y auditoría.
- Evitar registrar objetos completos cuando sea suficiente registrar identificadores.
- Incluir identificadores de correlación (UUID) para facilitar la trazabilidad entre servicios.
- Mantener consistencia en el formato de logs entre todas las tecnologías utilizadas.
- Validar periódicamente la correcta llegada de eventos al tablero corporativo.
- Evitar el uso de logs INFO, DEBUG y TRACE en ambientes productivos.

Cumplimiento

Todos los nuevos desarrollos, mantenimientos evolutivos y correctivos deberán cumplir estos lineamientos antes de ser promovidos a ambientes de pruebas o producción.

10. Lineamientos de Auditoría para Aplicaciones y Microservicios

10.1. Objetivo

Establecer los lineamientos para el registro de eventos de auditoría sobre las operaciones de negocio realizadas por los usuarios dentro de las aplicaciones institucionales, garantizando trazabilidad, control, seguimiento y capacidad de reconstrucción de cambios realizados sobre la información.

10.2. Alcance

La auditoría deberá registrar todas las operaciones relevantes efectuadas por usuarios internos, externos o procesos automáticos sobre entidades de negocio administradas por las aplicaciones.

La auditoría es independiente de los logs técnicos de la aplicación y tiene como propósito conocer:

- Quién realizó una acción.
- Cuándo la realizó.
- Sobre qué información se ejecutó.
- Qué información fue modificada.
- Cuál era el estado anterior.
- Cuál fue el nuevo estado.

10.3. Operaciones Permitidas

El campo TipoAuditoria únicamente podrá almacenar los siguientes valores:

Valor	Descripción
CREAR	Creación de un nuevo registro o entidad
CONSULTAR	Consulta o visualización de información
MODIFICAR	Actualización de información existente
ELIMINAR	Eliminación lógica o física de registros

No se permitirá el uso de valores distintos a los definidos interiormente.

Estructura de la Auditoría
Id
Identificador único de la auditoría generado por la base de datos.
Ejemplo:
125487
Aplicación

Nombre de la aplicación que genera el evento de auditoría.

Este valor deberá obtenerse desde el catálogo oficial de aplicaciones de Sucursal Digital y no deberá ser digitado manualmente por los desarrolladores.

Ejemplos:

safia
correo-masivo
certificados-en-linea
calendario-eventos

TipoAuditoria
Tipo de operación ejecutada.
Valores permitidos:
CREAR
CONSULTAR
MODIFICAR
ELIMINAR

Entidad
Nombre lógico de la entidad de negocio afectada.
Ejemplos:
Proceso
Usuario
Solicitud
Radicado
Certificado

Identificador
Objeto JSON que contiene los identificadores únicos de la entidad auditada.
Debe permitir identificar de forma inequívoca el registro afectado.
Ejemplo:

```
{
  "IdProceso":"2025-800-00361"
}
```

o

```
{
  "IdUsuario":"12345"
}
```

Programa
Nombre del caso de uso, servicio o funcionalidad que ejecutó la acción.
Ejemplos:
micro-safia
micro- expediente-digital

Componente
Nombre del módulo funcional o agregado de negocio responsable de la operación.
Ejemplos:
Proceso
Expediente
Usuario
Autenticación
Notificaciones
Fecha
Fecha y hora exacta en que ocurrió el evento auditado. Debe registrarse utilizando la hora oficial del servidor.
Ejemplo:
2025-11-27 08:48:15.903
Usuario
Usuario autenticado que ejecutó la operación.
Ejemplos:
usuario123
yquintana
Máquina
Dirección IP o nombre del equipo desde donde se originó la operación.
Ejemplos:
192.168.1.50
SSWV2-SDI03
ContenidoAnterior
Representa el estado de la información antes de ejecutar la operación.

Cuando debe utilizarse:

MODIFICAR

Debe almacenar la versión completa del registro antes del cambio.
Ejemplo:

```
{
  "Estado":"ACTIVO",
  "Nombre":"Proceso Ordinario"
}
```

ELIMINAR
Debe almacenar la última versión existente antes de eliminar el registro.

CREAR
Debe registrarse como NULL.

CONSULTAR
Debe registrarse como NULL.

ContenidoNuevo
Representa el estado de la información después de ejecutar la operación.

Cuando debe utilizarse:

CREAR
Debe almacenar el registro recién creado.
Ejemplo:

```
{
  "CodigoProceso":"2025-800-00361",
  "Estado":"ACTIVO"
}
```

MODIFICAR
Debe almacenar la versión final del registro después de aplicar los cambios.
Ejemplo:

```
{
  "CodigoProceso":"2025-800-00361",
  "Estado":"FINALIZADO"
}
```

ELIMINAR
Debe registrarse como NULL.
CONSULTAR
Debe registrarse como NULL.

10.4. Reglas para el Manejo de Contenidos

Los campos ContenidoAnterior y ContenidoNuevo deberán almacenarse en formato JSON.

Se recomienda incluir únicamente atributos de negocio relevantes para la trazabilidad.

No deberán registrarse:

* Contraseñas.
* Tokens.
* Secretos de aplicación.
* Archivos en Base64.
* Información financiera sensible.
* Datos personales sujetos a protección especial.
* Documentos completos o archivos binarios.

Cuando sea necesario registrar datos sensibles, estos deberán almacenarse enmascarados.

Ejemplo:

```
{
  "correo":"y******@dominio.com",
  "telefono":"300****123"
}
```

Ejemplos de Auditoría

CREAR

Campo	Valor
TipoAuditoria	CREAR
ContenidoAnterior	NULL
ContenidoNuevo	Registro creado

CONSULTAR

Campo	Valor
TipoAuditoria	CONSULTAR
ContenidoAnterior	NULL
ContenidoNuevo	NULL

MODIFICAR

Campo	Valor
TipoAuditoria	MODIFICAR
ContenidoAnterior	Estado antes del cambio
ContenidoNuevo	Estado después del cambio

ELIMINAR

Campo	Valor
TipoAuditoria	ELIMINAR
ContenidoAnterior	Último estado del registro
ContenidoNuevo	NULL

Buenas Prácticas

- Auditar únicamente operaciones de negocio.
- No utilizar la auditoría para reemplazar los logs técnicos.
- Mantener consistencia en los nombres de entidades y componentes.
- Registrar siempre el usuario responsable de la acción.
- Registrar los identificadores funcionales de la entidad auditada.
- Utilizar formato JSON para facilitar búsquedas y análisis posteriores.
- Evitar almacenar información sensible o protegida.
- Garantizar que las auditorías no puedan ser modificadas por usuarios finales.

11. Estructura del Código

Se refiere a la forma en que se organiza y distribuye el código fuente en un programa. Una buena estructura de código facilita la comprensión, el mantenimiento y la evolución del software motivo por el cual se aconseja:

- Tamaño máximo de las funciones (por ejemplo, no más de 100 líneas).
- Uso de comentarios (cuándo y cómo).
- Evitar duplicación de código (principio DRY - Don't Repeat Yourself).
- Variables quemadas
- Uso de tildes o acentos en los nombres de las clases, métodos, interfaces, paquetes.
- Las urls deben estar en un archivo externo que no sea envuelto por el compilador y sea fácil de editar por el área de infraestructura o en una tabla de la base de datos.
- Los nombres de los métodos deben tener como máximo 50 caracteres.
- Los nombres de las variables deben tener como máximo 20 caracteres.
- Los métodos deben tener como máximo 20 parámetros.
- Los principios **SOLID** son un conjunto de buenas prácticas de diseño orientado a objetos cuyo objetivo es crear software más mantenible, extensible y fácil de comprender.

  S — Single Responsibility Principle (SRP)
  Cada clase, módulo o función debe tener una única responsabilidad.
  O — Open/Closed Principle (OCP)
  Las clases deben estar abiertas para extensión, pero cerradas para modificación.
  L — Liskov Substitution Principle (LSP)
  Las clases hijas deben poder sustituir a sus clases padre sin alterar el comportamiento del sistema.
  I — Interface Segregation Principle (ISP)
  Las interfaces deben ser específicas, no genéricas.
  D — Dependency Inversion Principle (DIP)
  Los módulos de alto nivel no deben depender de módulos de bajo nivel; ambos deben depender de abstracciones.

- El principio **KISS** de diseño que establece que la mayoría de los sistemas funcionan mejor cuando se mantienen simples en lugar de complejos.

11.1. Control de Versiones

Es un sistema que permite llevar un registro de los cambios realizados en un proyecto o en archivos a lo largo del tiempo. Es especialmente útil en el desarrollo de software ejecutado por la fábrica de desarrollo, ya que facilita la

colaboración entre varios desarrolladores, la gestión de diferentes versiones del código y la recuperación de versiones anteriores en caso de errores o problemas.

11.2. Repositorio

Es el espacio donde se almacena y gestiona el código fuente desarrollado por la fábrica, archivos o datos de un proyecto. En el contexto de desarrollo de software de la entidad, el repositorio contiene todo el historial de cambios, la estructura del proyecto y los archivos necesarios para que el software funcione correctamente.

- URL del repositorio.
- Plataforma: azure Devops, etc.

Los repositorios institucionales deberán mantenerse como privados salvo autorización expresa y documentada del responsable del activo.

Se deberá realizar una revisión periódica de los repositorios públicos para verificar:

- Exposición accidental de código fuente.
- Publicación no autorizada de información institucional.
- Divulgación de configuraciones sensibles.
- Exposición de documentación técnica restringida.

11.3. Ramas

Son una forma de gestionar el desarrollo de un proyecto de software mediante la separación de cambios y características en líneas de trabajo independientes. Al trabajar con ramas, puedes desarrollar nuevas características, corregir errores o realizar mejoras sin afectar el código principal del proyecto.

- Estándar de uso de ramas: main, develop, feature/nombre-funcionalidad.
- Proceso para integrar cambios (pull requests, code reviews).

11.4. Commits

Es una acción en sistemas de control de versiones (como Git) que guarda y registra los cambios realizados en los archivos de un repositorio. Cada commit representa un "punto de control" o una instantánea del proyecto en un momento

determinado. Los commits permiten llevar un registro detallado de todo el historial de cambios en un proyecto, lo que facilita el seguimiento, la colaboración y la reversión de cambios si es necesario.

- Mensajes de commit claros y descriptivos.
- Formato de Commit:
  - Tipo: Indica la categoría del cambio. Los más comunes son:
    - **feat**: Una nueva funcionalidad.
    - **fix**: Corrección de un bug.
    - **docs**: Cambios relacionados con la documentación.
    - **style**: Cambios de formato o estilo (sin afectar lógica).
    - **refactor**: Cambios en el código sin agregar funcionalidades ni corregir bugs.
    - **test**: Agregar o modificar pruebas.
    - **chore**: Cambios en tareas de construcción, configuraciones o herramientas auxiliares
  - Área: (opcional) Indica el módulo, archivo o área del código afectada, Ejemplo: autenticación, usuario, UI.
  - Resumen Corto: Descripción breve (máximo 50 caracteres). Usa un verbo en imperativo presente (e.g., agrega, ajusta, actualiza) y en español.

  Ejemplos:

  - **feat(autenticacion)**: agrega funcionalidad en el inicio de sesión.
  - **feat(db_sociedades)**: agrega tabla usuarios con campos nombre y correo.
  - **fix(autenticacion)**: resuelve la excepción de puntero nulo al iniciar sesión.
  - **fix(db_sociedades)**: corrige relaciones entre persona y tipo de identificación.
  - **docs(readme)**: actualiza la sección de instalación.
  - **docs(readme)**: agrega instrucciones para ejecutar migraciones de la base de datos db_sociedades.
  - **style(UI)**: ajusta el espaciado en el formulario de registro.
  - **style(sql)**: ajusta el formato de consultas SQL para mayor legibilidad.

  - **style(UI)**: ajusta la disposición de columnas en la vista de resultados
  - **Refactor(db_sociedades)**: migra a una nueva versión del esquema.
  - **Refactor(db_sociedades)**: refactoriza el esquema para soportar múltiples direcciones por usuario.
  - **test(persona)**: agrega pruebas unitarias para validación de persona.
  - **test(db_sociedades)**: agrega pruebas unitarias para verificar consultas de usuarios activos
  - **chore(JDK)**: actualiza la versión de java a la última estable.
  - **chore(db_sociedades)**: elimina la columna obsoleta descripción de la tabla sociedades.
- Buenas Prácticas
  - **Se debe ser claro y conciso**: Utilizar oraciones cortas y directas.
  - **Usar el imperativo presente**: "agrega" en lugar de "agregada" o "agregando", otros ejemplos: ajusta, corrige, refactoriza, actualiza, elimina.
  - **Agrupa cambios relacionados**: No combines cambios sin relación en un solo commit.
  - **Respeta el límite de caracteres**: Máximo 50 caracteres para el título, 72 caracteres por línea en el cuerpo.
  - **Subir cambios diariamente** o al finalizar un bloque de trabajo significativo.
  - Cada commit debe ser **funcional**, es decir:
    - Compila sin errores.
    - No rompe flujo existente.
    - Pasa pruebas básicas (unitarias o manuales relevantes).
  - Evitar commits con código incompleto, comentado sin propósito o que deje el proyecto en un estado inconsistente.
  - Beneficios:
    - Facilita detectar errores recientes.
    - Minimiza conflictos de merge.
    - Asegura continuidad del proyecto para otros equipos.
- Evitar commits masivos

a. Los cambios deben ser atómicos: una sola responsabilidad por commit.
b. Evitar subir cientos de archivos en un solo commit, salvo casos de migración controlada.
c. Esto facilita:
    1. Revisiones de código.
    2. Reversiones seguras.
    3. Aislar defectos.

- Manejo apropiado de .gitignore

  Se deben excluir:

  a. Archivos generados automáticamente (build, dist, target, class).
  b. Credenciales y configuraciones locales.
  c. Dependencias de terceros (node_modules, packages, .jar, .ddl).
  d. Mantener un .gitignore limpio y consistente en todos los proyectos.

- Sin credenciales en el repositorio

  Nunca subir:

  a. Passwords
  b. Tokens
  c. Certificados
  d. Cadenas de conexión
  e. Usar archivos de configuración separados, variables de entorno o servicios como Key Vault/Secrets Manager.

- Documentar cambios significativos

  Ante modificaciones estructurales:

  a. Actualizar README.
  b. Actualizar arquitectura o diagramas si aplica.
  c. Esto ayuda a nuevos desarrolladores a integrarse rápidamente.

- Resolver conflictos de forma consciente

  Antes de hacer merge:

  a. Actualizar la rama local (git pull o git fetch + rebase).
  b. Revisar conflictos con criterio funcional.

c. Evitar eliminar código sin entender el impacto.

12. Base de Datos

Es un sistema organizado para almacenar, gestionar y recuperar datos de manera eficiente. Las bases de datos son fundamentales para el almacenamiento de información en aplicaciones de software, ya que permiten organizar grandes cantidades de datos de manera estructurada y accesible. El cual se utilizará en SQL Server.

12.1. Nombre de la Base de Datos

Se debe especificar un nombre que esté relacionado con la aplicación o el nombre del proyecto para el cual fue desarrollada dicha base de datos. Hay que tener en cuenta que si el nombre de la aplicación o proyecto, para el cual se va a desarrollar la base de datos, es muy largo, se prefiere utilizar el alias de la aplicación.

- Si el nombre es definido en siglas, el nombre se define en mayúscula sostenida.
- Se el nombre es de una palabra o palabra compuesta, la primera letra de cada palabra va en mayúscula y el resto en minúscula (nomenclatura pascal).
- El nombre de la base de datos está limitado a 32 caracteres.
- Puede ir en singular o plural, dependiendo del caso en el que el negocio requiera de expresiones en plural o singular.
- Deben nombrarse usando notación pascal.
- El nombre termina con la abreviación del ambiente para el cual se crea la base de datos.
- Nomenclatura: nombre del proyecto + abreviatura del ambiente

Ejemplo área de pruebas: CalidadDatosQA
Ejemplo desarrollo: CalidadDatosDEV
Ejemplo producción: CalidadDatosPRD

Ejemplo de nombres físicos de archivos de BD:
Archivo MDF: CalidadDatosQAData.MDF
Achivo LDF: CalidadDatosQALog.LDF

12.2. Nombre de las instancias de Bases de Datos

Para nombrar una instancia de base de datos se debe establecer un nombre que defina una particularidad del modelo. Se establecen las siguientes condiciones:

- El nombre debe ir en singular.
- Se debe utilizar la nomenclatura pascal.
- Los nombres de instancia no pueden contener los términos reservador por el lenguaje estructurado de consulta SQL (Default, MSSQLServer, ETC).
- Los nombres de instancias están limitados a 32 caracteres..
- El primer carácter del nombre de la instancia debe ser una letra o un carácter de subrayado (_). Las letras aceptables son las que define el estándar Unicode 2.0, que incluye los caracteres latinos a-z y A-Z, además de caracteres de letras de otros idiomas.
- Los siguientes caracteres pueden ser letras definidas por el estándar Unicode 2.0, números enteros del alfabeto latín básico y de otros alfabetos nacionales, sin un carácter de subrayado (_).
- En los nombres de instancia no se permiten espacios incrustados de otros caracteres especiales, únicamente se permiten caracteres latinos a-z y A-Z y números enteros

12.3. Nombre de esquemas

Se debe especificar un nombre que esté relacionado con el área del negocio que se esté trabajando dentro de la aplicación o proyecto, cuando esta pueda contener varias áreas o módulos dentro de la aplicación pero que estén claramente identificadas. Por ejemplo, en el caso del proyecto de calidad de datos tenemos identificados varios dominios en donde se clasifican las diferentes tablas que se manejan dentro del modelo desarrollado para el proyecto en los cuales se encuentran: personas naturales y jurídicas, administradores,

localización, sociedades supervisadas, y auditoria.

Ejemplo: Sirfin.ActualizaControl, donde el esquema es Sirfin sistema al que pertenece el proceso y Actualiza control el nombre de la tabla o vista según sea el caso.

Estandarización de Esquemas

Definir una estructura estándar de esquemas para todas las aplicaciones y microservicios, con el fin de:

- Garantizar consistencia en el modelo de datos
- Facilitar mantenimiento y escalabilidad
- Permitir reutilización de componentes
- Mejorar la gobernanza de datos

Definición de Esquemas

Esquema de Negocio (<modulo> o negocio)

Contiene las **entidades operativas del sistema**.
Características:

- Datos transaccionales
- Alta frecuencia de escritura
- Representan procesos del negocio
- **Ejemplos de tablas:**
- Evento
- Alerta
- AlertasGeneradas
- Radicacion

Esquema de Parametría (Parametria)

Contiene **catálogos controlados y reutilizables.**
Características:

- Valores relativamente estáticos
- Compartidos entre módulos
- Definen dominios de datos

Ejemplos de tablas:

- TipoAlerta
- TipoEvento
- TipoTramite
- Parametro
- EstadoAlerta

Esquema de Configuración (Configuracion)

Contiene **la configuración dinámica del sistema.**

Características:

- Puede cambiar sin despliegue
- Define comportamiento del sistema
- Puede variar por ambiente o cliente

Ejemplos de tablas:

- ConfiguracionSistema

La estandarización evita el uso indiscriminado del esquema dbo, reduce duplicidad de estructuras y permite que todas las aplicaciones mantengan una arquitectura de persistencia alineada a las prácticas corporativas y de microservicios.

12.4. Convención General

Negocio o dominio específico (Radicacion, Eventos, Alerta)	Tablas principales del negocio
Parametría	Catálogos funcionales de negocio
Configuración	Configuración técnica o transversal
Seguridad	Usuarios, roles, permisos, autenticación
Operación	Trazabilidad, procesos, saga, jobs
Auditoria	Auditoría funcional o técnica
Integración	Integraciones externas, colas, APIs

Definición de Esquemas

Esquema	Tipo	Caso de	Ejemplos

		uso	
Radicación	Negocio	Tablas propias del proceso de radicación	Radicado, DocumentoRadicado
Eventos	Negocio	Gestión funcional de eventos	Evento, CalendarioEvento
Alerta	Negocio	Gestión de alertas de Safia	Alerta, AlertaGenerada
Parametría	Catálogo negocio	Información configurable funcional por usuarios	Ciudad, Departamento, TipoDocumento, Criterio, EstadoRegistro(activo, inactivo)
Configuración	Técnico	Configuración interna del sistema	EstadoProceso(OK,ERROR,REINTENTO,PROCESADO CANCELADO), TipoAlerta, ConfiguracionCorreo
Seguridad	Seguridad	Autenticación y autorización	Usuario, Rol, Permiso, UsuarioRol
Operación	Operación técnica	Trazabilidad, saga, jobs, procesos internos	TrazabilidadMicroservicio, SagaTransaccion
Auditoria	Auditoría	Históricos y seguimiento funcional	AuditoriaEvento, AuditoriaCambioEstado
Integración	Integración	Integración con sistemas externos	ColaPendiente, EventoIntegracion, Webhook

Nombramiento de Tablas

Las tablas deben nombrarse:

- En plural

- En español
- Sin utilizar espacios en blanco
- Si el nombre es compuesto solo la última palabra debe ir en plural. Por ejemplo: ClaseSociedades
- Deben nombrarse usando notación pascal.
- La longitud máxima de los nombres de las tablas es de 50 caracteres.
- Todas las tablas tienen un índice único generado por el sistema por default newid()Cluster
- Todos los índices de referencia deben ir al comienzo de la tabla
- Todas las tablas deben tener llave primaria (única) de información
- Todas las tablas deben tener los campos FechaCreacion por defecto getdate() y FechaModificacion.
- Las tablas de tipo transacción deben tener un campo denominado EstadoRegistro se utiliza para marcar el registro como anulado y evitar borrado y Prioridad y se utilizan para llevar el histórico en la misma tabla.
- Las Tablas tipo Paramétrica utilizan el EstadoRegistro para activar e inactivar el registro.

Ejemplo tablas nombre compuesto:

PersonaNaturalJuridica, SociedadSupervisada

Ejemplo tablas nombre simple: País, Ciudad

Nombramiento de atributos (Nombre Columnas Tablas)

Para el nombramiento de los atributos de las tablas (columnas) se deben utilizar nombres significativos para el negocio con respecto al dato que se va a almacenar en dicha columna. Lo anterior con el fin de permitir saber de primera mano los datos almacenados en cada una de las columnas sin tener la necesidad de pedir la asesoría o acompañamiento de un experto en el modelo desarrollado.

- La longitud máxima de los nombres de los atributos es de 50 caracteres.
- Los nombres se escriben en singular y en español
- Deben nombrarse usando notación pascal.
- Se debe usar el atributo **Description** para establecer el metadato que describe el campo.

- Si se requiere un campo serial, se define en la propiedad de la columna **indentity specification**, no se debe utilizar en lo posible la opción de secuencias.

Se debe escribir el nombre sin utilizar en la medida de lo posible abreviaciones en el nombre y al final del nombre se pone el sufijo "**Id**"; adicional a esto, se debe definir el nombre del atributo en singular.

Ejemplo: PersonaNaturalJuridicaId

Nombramiento de restricciones (constraints)

Dependiendo de la naturaleza de la restricción se define el estándar de nombramiento de las restricciones definidas a nivel de base de datos.

- La longitud máxima de los nombres de los constraints es de 50 caracteres.
- Los nombres se escriben en singular y en español
- Deben nombrarse usando notación pascal.

Restricción de llave primaria
Se debe nombrar la llave primaria iniciando con la cadena "PK_" seguido del nombre de la tabla donde se define este constraint.
Ejemplo: PK_PersonaNaturalJuridica

Restricción de llave foránea
Se debe nombrar la llave foránea iniciando con la cadena "FK_" seguido de la combinación del nombre de la tabla donde donde se encuentra la llave primaria, más el nombre de la tabla donde se define la llave foránea.
Ejemplo: FK_GeneroPersonaNaturalJuridica

Restricción de llave única
Se debe nombrar la llave única iniciando con la cadena "UQ_" seguido del nombre de la tabla donde se define este constraint escrito en singular.
Ejemplo: UQ_Persona

Restricción de tipo check
Se debe nombrar la restricción de tipo check iniciando con la cadena "CK_" seguido del nombre del atributo de la tabla donde se define este constraint.
Ejemplo: CK_TipoPersona

Nombramiento de índices

Se debe nombrar el índice iniciando con la cadena "IDX_" seguido del nombre del atributo de la tabla donde se define este o la combinación de los atributos que hacen parte de este cuando sea un índice compuesto y la longitud máxima de los nombres de los índices es de 100 caracteres.

Nombramiento de procedimientos almacenados

En caso sea necesario trabajar con este tipo de STORED PROCEDURES, deberán ser creados de la siguiente forma:

Sintaxis

```
Create Proc sp_SelTabla
@X tinyint,
@Y char(3)
WITH RECOMPILE
as
if @X=1
        Select * from Tablas
Else
        Select * from Tablas where codigo = @Y
```

No se deben usar procedimientos encriptados. Todo aplicativo deberá evitar usar cursores definidos en SQL SERVER, dado que éstos consumen muchos recursos. En la mayoría de los casos se pueden transformar los cursores a Transact-SQL.

Sintaxis

- Identificador, nombre de stored procedure, y parámetros
- Declaración Variables locales
- Mayúsculas y minúsculas
- Sentencias SQL

- Palabras del lenguaje SQL, y funciones de sistema en MAYUSCULAS, columnas y otras variables en Mayúsculas.
- Sentencias legibles e indentadas (cada cláusula SQL en una línea nueva)

Sintaxis

```
CREATE PROCEDURE sp_InsSociedad(
    @pCodEmpresa int,
    @pRazonSocial varchar(50))
AS
<Declaración de variables>
<Sentencias SQL>
```

Nombramiento de funciones

Una función definida por el usuario es una rutina de Transact-SQL o Common Language Runtime (CLR) que acepta parámetros, realiza una acción, como un cálculo complejo, y devuelve el resultado de esa acción como un valor. El valor devuelto puede ser un valor escalar (único) o una tabla. Utilice esta instrucción para crear una rutina reutilizable que se pueda utilizar de estas formas:

- En instrucciones Transact-SQL como SELECT
- En las aplicaciones que llaman a la función
- En la definición de otra función definida por el usuario
- Para parametrizar una vista o mejorar la funcionalidad de una vista indizada
- Para definir una columna en una tabla
- Para definir una restricción CHECK en una columna
- Para reemplazar un procedimiento almacenado

Nomenclatura

- Usar notación Pascal
- La longitud máxima de los nombres de las funciones es de 50 caracteres.
- Nomenclatura propuesta: “Fn” + Acción + ”_” + Entidad involucrada en la acción

Fn: Abreviatura para significar que es una Función

Las acciones pueden ser:

- Ins: Inserción
- Upd: Actualización
- Del: Borrado
- Sel: Consulta
- Grl: Cualquier combinación de los tipos anteriores u otro tipo de procesos.

Ejemplos: FN_Ins_Sociedad

Nombramiento de Vistas

Crea una tabla virtual cuyo contenido (columnas y filas) se define mediante una consulta. Utilice esta instrucción para crear una vista de los datos de una o varias tablas de la base de datos. Por ejemplo, una vista se puede utilizar para lo siguiente:

- Para centrar, simplificar y personalizar la percepción de la base de datos para cada usuario.
- Como mecanismo de seguridad, que permite a los usuarios obtener acceso a los datos por medio de la vista, pero no les conceden el permiso de obtener acceso directo a las tablas base subyacente de la vista.
- Para proporcionar una interfaz compatible con versiones anteriores para emular una tabla cuyo esquema ha cambiado.

Nomenclatura

Las vistas deben nombrarse con la misma notación definida para nombrar tablas, pero usando el prefijo "vw_".

- En singular.
- En español.
- La longitud máxima de los nombres de las vistas es de 50 caracteres.
- Sin utilizar espacios en blanco.
- Deben nombrarse usando notación pascal.
- Debe iniciar con el nombre del esquema.

Ejemplo: CalidadDatos.vw_direcciones

Nombramiento de Triggers

Crea un disparador DML, DDL o de logon. Un disparador es una clase especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce un evento en el servidor de bases de datos. Los disparadores DML se ejecutan cuando un usuario intenta modificar datos mediante un evento de lenguaje de manipulación de datos (DML). Los eventos DML son instrucciones INSERT, UPDATE o DELETE de una tabla o vista. Estos desencadenadores se activan cuando se desencadena cualquier evento válido, con independencia de que las filas de la tabla se vean o no afectadas.

Nomenclatura

Nomenclatura propuesta: "trg" + Acción + "_" + Entidad involucrada en la acción

- Usar notación Pascal.
- La longitud máxima de los nombres de los triggers es de 50 caracteres.

trg: Abreviatura para significar que es un Trigger

Las acciones pueden ser:

- Ins: Inserción
- Upd: Actualización
- Del: Borrado
- Sel: Consulta
- Grl: Cualquier combinación de los tipos anteriores u otro tipo de procesos.

Ejemplos: trg_Ins_Sociedad

Reglas de programación

El nombramiento de las variables declaradas en los procedimientos almacenados, funciones y triggers deben seguir el estándar definido en la notación pascal.

Las palabras reservadas (instrucciones propias del SQL) siempre deben escribirse en mayúscula sostenida en todos los scripts generados para: crear tablas, vistas, procedimientos, funciones, índices, triggers, entre otros.

13. Pruebas

13.1. Tipos de Pruebas

Unitarias: son un tipo de prueba de software que se enfocan en verificar el comportamiento de una unidad o componente individual de un sistema, de manera aislada. Este componente puede ser una función, un método o una clase. El objetivo de las pruebas unitarias es asegurar que cada unidad del código funcione correctamente, en su comportamiento más básico y sin depender de otras partes del sistema. El patrón para utilizar en los desarrollos de la superintendencia de sociedades es el patrón triple A (**Arrange – Act – Assert**) es una convención ampliamente utilizada para estructurar pruebas unitarias de forma clara, ordenada y fácil de entender. Su objetivo es separar explícitamente cada fase de la prueba para mejorar la legibilidad, el mantenimiento y la coherencia del código de prueba.

Arrange (Preparar
En esta fase se configura el escenario necesario para ejecutar la prueba
Act (Actuar)
Es la ejecución del comportamiento a probar.
Assert (Afrimar / Verificar)
Se verifica el resultado de la acción realizada.

Integración: son un tipo de prueba de software que se enfocan en verificar la interacción entre múltiples componentes o unidades de un sistema. El objetivo de estas pruebas es asegurar que los diferentes módulos o partes del sistema trabajen correctamente cuando se combinan, identificando posibles problemas en las interfaces y en la comunicación entre componentes.

Aceptación: son un tipo de prueba de software que se realizan para verificar si el sistema cumple con los **requisitos del usuario** y si está listo para ser entregado o implementado en un entorno de producción. Estas pruebas validan que el producto o sistema satisface las expectativas y necesidades del cliente o usuario final, y aseguran que el software está listo para su uso real.

Pruebas de carga: son un tipo de prueba de rendimiento de software que se realizan para evaluar cómo un sistema o aplicación maneja un volumen específico de usuarios o transacciones simultáneas. El objetivo principal es verificar que el sistema pueda manejar la carga esperada en condiciones normales de uso, sin degradarse en rendimiento o provocar fallos.

Pruebas en Postman: Son un tipo de pruebas de API (Interfaz de Programación de Aplicaciones) que se realizan utilizando la herramienta Postman. Esta herramienta permite crear, ejecutar y automatizar pruebas de solicitudes HTTP para verificar que las API funcionen correctamente, cumpliendo con los requisitos especificados. Las pruebas en Postman son especialmente útiles para probar servicios web y APIs RESTful, asegurando que respondan adecuadamente a las solicitudes realizadas. Se debe compartir las collection (entregable).

13.2. Cobertura de Pruebas

Es una métrica utilizada para medir la cantidad de código o funcionalidad que ha sido ejecutada y verificada durante el proceso de pruebas de software. En otras palabras, la cobertura de pruebas indica el porcentaje del código fuente de una aplicación que ha sido probado por medio de pruebas, ya sean unitarias, de integración, de aceptación, de carga entre otras y el Objetivo de cobertura mínima del código. Para la superintendencia de sociedades a nivel de back debe ser solo de a nivel de la capa de negocio sobre el 90% de la cobertura y a nivel de front del 80%.

13.3. Herramientas de Pruebas

Framework de pruebas: es una estructura que proporciona un conjunto de herramientas y convenciones para facilitar la creación, ejecución y gestión de pruebas automatizadas. Los frameworks de pruebas ayudan a los desarrolladores a escribir pruebas de manera más eficiente, estructurada y consistente, ofreciendo funcionalidades como aserciones, organización de pruebas, ejecución de casos de prueba y generación de reportes.

Entre los más conocidos esta:

Jest: Es un framework de pruebas para JavaScript, desarrollado y mantenido por Facebook. Es ampliamente utilizado para realizar pruebas unitarias, de integración y de aceptación en aplicaciones que usan React, pero también se puede utilizar en otros proyectos JavaScript.

PyTest: Es un framework de pruebas para Python que facilita la creación de pruebas automatizadas. Es muy popular debido a su simplicidad, flexibilidad y la gran cantidad de características avanzadas que ofrece.

Mocha: Es un framework de pruebas (testing) para JavaScript, diseñado para facilitar la escritura y ejecución de pruebas unitarias y de integración. Es muy popular en el ecosistema de desarrollo de aplicaciones, especialmente para proyectos Node.js, aunque también puede usarse en el navegador.

14. Entorno de Desarrollo

14.1. Configuración

14.1.1. Java

Sistemas Operativos Soportados

Para el entorno de desarrollo de java se pueden utilizar los mismos Sistemas operativos (S.O) Windows, Linux y macOS. Java es conocido por su característica "Write Once, Run Anywhere" gracias a la JVM. Funciona en cualquier sistema operativo que tenga una implementación compatible de la JVM.

Dependencias Principales

JDK (Java Development Kit): Es necesario tener instalado el JDK para compilar y ejecutar aplicaciones Java. Existen varias implementaciones de JDK:

- **Oracle JDK**: La implementación oficial de Oracle.
- **OpenJDK**: Es una implementación de código abierto del JDK.

- **AdoptOpenJDK** (ahora llamado Adoptium): Una distribución comunitaria de OpenJDK.
- **GraalVM**: Una implementación avanzada de la JVM con características adicionales.

IDE recomendados:

- **IntelliJ IDEA**: Muy popular para Java, con soporte completo para herramientas modernas y lenguajes relacionados (Kotlin, Scala, etc.).
- **Eclipse**: Uno de los IDEs más antiguos y robustos para Java, conocido por su extensibilidad.
- **NetBeans**: Otro IDE que soporta Java y está orientado a desarrolladores de aplicaciones empresariales.
- **VS Code**: Usado con la extensión de Java para aquellos que prefieren editores de texto más ligeros.

14.1.2. C#

Sistemas Operativos Soportados

Para el entorno de desarrollo de C# es indispensable contar con alguno de los siguientes sistemas operativos:

- **Windows**: Es la plataforma principal para C# debido a su integración nativa con el ecosistema de Microsoft.

- **Linux y macOS**: Con la llegada de .NET Core y luego .NET 5/6/7+, C# ahora es completamente compatible con sistemas operativos basados en Linux y macOS, lo que permite la ejecución y desarrollo de aplicaciones multiplataforma.

Dependencias Principales

- **.NET SDK**: Necesitarás el SDK de .NET para desarrollar aplicaciones en C#. Esto incluye las herramientas de compilación y las bibliotecas básicas. Puedes descargar el SDK desde la página oficial de Microsoft.
- **.NET Core**: Para versiones anteriores a .NET 5.

- **.NET 5/6/7**: Para versiones más recientes que son completamente multiplataforma.
- **Visual Studio (principal IDE)**: Ofrece soporte completo para el desarrollo de C# en Windows, y también tiene una versión para macOS llamada Visual Studio for Mac.
- Para Linux, algunos usuarios prefieren usar VS Code con las extensiones de C#.
- Herramientas de línea de comandos de .NET: Permiten crear, compilar, ejecutar y administrar proyectos C# directamente desde la terminal, con herramientas como dotnet build, dotnet run, etc.

Característica	C#	Java
Sistemas operativos	Windows, Linux, macOS (con .NET Core/5+)	Windows, Linux, macOS (con JVM)
IDE principales	Visual Studio, Visual Studio for Mac, VS Code	IntelliJ IDEA, Eclipse, NetBeans, VS Code
Herramientas de construcción	MSBuild, NuGet, dotnet CLI	Maven, Gradle
Compatibilidad	.NET Core/5+ (multiplataforma)	Multiplataforma a través de la JVM
Gestión de dependencias	NuGet (paquetes)	Maven, Gradle (dependencias externas)
Uso común	Aplicaciones de escritorio, móviles (Xamarin), servicios web	Aplicaciones empresariales, servicios web, aplicaciones móviles (Android)

14.2. Requisitos para desarrollar

14.2.1. Java

Los requisitos de hardware para desarrollar en Java son similares a los de C#, aunque pueden variar dependiendo del tamaño y la complejidad del proyecto.

Procesador (CPU)

- **Mínimo**: Procesador de 1.8 GHz o superior (dual-core es adecuado para muchos proyectos pequeños).
- **Recomendado**: Procesador de múltiples núcleos (Intel i5/i7 o AMD Ryzen) para manejar compilaciones y ejecución de proyectos más grandes.

Memoria RAM

- **Mínimo**: 4 GB de RAM.
- **Recomendado**: 8 GB o más de RAM si se usan herramientas pesadas o si se desarrollan aplicaciones grandes o con múltiples dependencias.

Espacio en Disco Duro

- **Mínimo**: 2 GB de espacio libre para el JDK y herramientas básicas.
- **Recomendado**: 10 GB o más si planeas trabajar con aplicaciones empresariales grandes, frameworks como Spring, bases de datos locales, y otros recursos.

Tarjeta Gráfica

- **Mínimo**: Los gráficos integrados son suficientes para Java.
- **Recomendado**: No se requiere una GPU dedicada, a menos que trabajes con aplicaciones gráficas intensivas.

14.2.2. C#

Para el entorno de desarrollo de C# no se requieren especificaciones extremadamente altas, pero si lo suficientemente capaces de poder compilar y/o ejecutar un proyecto.

Procesador (CPU)

- **Mínimo**: Procesador de 1.8 GHz o superior (Dual-core o más recomendado).
- **Recomendado**: Procesador de múltiples núcleos (Intel i5/i7 o AMD Ryzen), para un rendimiento más fluido al compilar y ejecutar grandes proyectos.

Memoria RAM

- **Mínimo**: 4 GB de RAM.
- **Recomendado**: 8 GB de RAM o más para un desarrollo más eficiente, especialmente si usas aplicaciones más pesadas o múltiples instancias de IDEs y simuladores/emuladores.

Espacio en Disco Duro

- **Mínimo**: 4 GB de espacio libre en disco para la instalación de herramientas básicas.
- **Recomendado**: 10 GB o más de espacio libre si planeas usar múltiples herramientas y bibliotecas de NuGet.

Tarjeta Gráfica

- **Mínimo**: Gráficos integrados (No es esencial para la mayoría de los proyectos, pero una GPU dedicada puede mejorar la experiencia visual de aplicaciones).
- **Recomendado**: No es necesario, salvo si trabajas con aplicaciones que requieren gráficos 3D o procesamiento pesado (por ejemplo, videojuegos o software de diseño).

Requisito	C# (con .NET)	Java (con JDK)
Sistema Operativo	Windows, Linux, macOS (con .NET Core/5+)	Windows, Linux, macOS (con JVM)
CPU	1.8 GHz mínimo, multicore recomendado	1.8 GHz mínimo, multicore recomendado
RAM	4 GB mínimo, 8 GB recomendado	4 GB mínimo, 8 GB recomendado
Espacio en Disco	4 GB mínimo, 10 GB recomendado	2 GB mínimo, 10 GB recomendado
IDE	Visual Studio, VS Code, Rider	IntelliJ IDEA, Eclipse, NetBeans, VS Code
JDK/SDK	.NET SDK (Core/5+)	JDK (Oracle JDK, OpenJDK, Adoptium, etc.)
Herramientas de construcción	MSBuild, NuGet, dotnet CLI	Maven, Gradle

14.3. Entornos

Es **Recomendado** en el ciclo de vida del desarrollo de software, contar con diferentes **entornos de desarrollo, pruebas y producción**. Ya que cada uno tiene un propósito específico y contribuye a asegurar la calidad y estabilidad de la aplicación antes de que llegue a los usuarios finales.

Desarrollo: El entorno de desarrollo es el lugar donde los desarrolladores escriben, prueban y depuran el código. Este entorno está optimizado para permitir la edición rápida del código y la integración continua durante el proceso de creación del software.

Pruebas: El entorno de pruebas (a veces llamado QA, test o staging) es donde se realiza la verificación y validación de que el software cumple con los requisitos especificados y funciona correctamente bajo condiciones que simulan el entorno de producción.

Producción: El entorno de producción es el lugar donde la aplicación finalmente es utilizada por los usuarios finales. Es el entorno real donde la aplicación se ejecuta y está disponible para los usuarios en todo momento.

14.4. Herramientas

Entorno de desarrollo es un conjunto de herramientas, software y configuraciones que los programadores utilizan para desarrollar, probar y depurar aplicaciones de manera eficiente. Es el espacio en el que los desarrolladores escriben, prueban y mantienen su código antes de que el software sea lanzado a producción.

14.4.1. Java (Editores de código recomendados)

- Spring Tool Suite (STS) es un IDE especializado basado en Eclipse, diseñado específicamente para el desarrollo de aplicaciones Java usando el framework Spring. Fue creado por VMware/Pivotal para facilitar la vida de los desarrolladores que trabajan con el ecosistema Spring.
- Como en la superintendencia se trabaja con Spring Boot, Spring MVC, Spring Data, etc., STS es una de las mejores herramientas disponibles.

Características Principales:

- Basado en Eclipse, por lo que incluye toda su potencia y extensibilidad.
- Incluye soporte nativo para:
- Spring Boot Webx flux
- Spring Boot
- Spring MVC
- Spring Security
- Spring Data
- Thymeleaf, entre otros.
- Herramientas integradas para crear, ejecutar y depurar aplicaciones Spring Boot de forma sencilla.
- Integración con Maven y Gradle.
- Explorador de Beans de Spring (para ver las dependencias de tu aplicación).
- Asistentes y wizards para generar proyectos Spring con estructura predefinida.
- Incluye acceso directo al Spring Initializr para crear nuevos proyectos Spring fácilmente.

14.4.2. C# (Editores de código recomendados)

Visual Studio

Visual Studio es un Entorno de Desarrollo Integrado (IDE) completo desarrollado por Microsoft. Está diseñado principalmente para el desarrollo en C#, .NET, C++, y otros lenguajes compatibles, especialmente para proyectos grandes o complejos.

Características Principales:

- Entorno robusto y profesional para desarrollo de software.
- Soporte completo para .NET, C#, ASP.NET, Xamarin, C++, VB.NET, entre otros.
- Incluye herramientas integradas como:
- Depurador avanzado (debugger).
- Editor de formularios (Windows Forms, WPF).
- Diseñadores de UI.
- Pruebas unitarias integradas.
- Análisis de código y perfiles de rendimiento.
- Integración directa con Azure, SQL Server, Git y más.

- Soporte para desarrollo de aplicaciones de escritorio, web, móviles, videojuegos (con Unity), y servicios en la nube.

Visual Studio Code

Visual Studio Code es un editor de código fuente ligero y multiplataforma también desarrollado por Microsoft. Aunque no es un IDE completo por sí solo, se puede extender con extensiones para adaptarse a casi cualquier lenguaje o tecnología.

Características Principales:

- Editor muy rápido y personalizable.
- Soporte para cientos de lenguajes mediante extensiones (C#, Java, Python, JavaScript, Go, PHP, etc.).
- Terminal integrada, Git integrado, y vista dividida.
- Extensiones para depuración, linting, IntelliSense, y snippets.
- Compatible con contenedores Docker, WSL (Windows Subsystem for Linux), y desarrollo remoto.
- Bajo consumo de recursos en comparación con IDEs tradicionales.

14.5. Herramientas de Integración

En la superintendencia se utiliza **Azure DevOps** el cual es un conjunto de herramientas y servicios en la nube proporcionado por Microsoft, diseñado para gestionar todo el ciclo de vida del desarrollo de software. Permitiendo a la fábrica de desarrollo de la entidad planificar, desarrollar, probar, entregar y monitorear software de forma colaborativa y eficiente, utilizando prácticas de DevOps.

15. Seguridad

En la superintendencia se utiliza https con oauth 2.0 con token (JWT) cuya fuente de datos o repositorio de usuarios es una base de datos relacional y el LDAP y cada aplicación maneja la autorización con la invocación de un servicio REST desarrollado en la capa de Interoperabilidad. Se debe seguir el estándar owasp top 10 en sus últimas versiones.

15.1. Normas para Manejar Credenciales y Permisos

Uso de Credenciales

- No compartir credenciales: Cada usuario debe tener sus propias credenciales. Nada de compartir contraseñas o accesos.
- Contraseñas fuertes: Usar contraseñas complejas (mínimo 12 caracteres, con letras, números y símbolos).
- No reutilizar contraseñas: Especialmente entre cuentas personales y profesionales.
- Cambios periódicos: Cambiar contraseñas regularmente o tras una sospecha de filtración.

Almacenamiento de credenciales

- Nunca en texto plano: No guardar contraseñas en archivos .txt, correos, notas o código fuente.
- Variables de entorno o servicios secretos: En desarrollo, almacenar claves/API keys en servicios como Vault, AWS Secrets Manager, o variables de entorno.

Asignación de permisos

- Principio de mínimo privilegio: Dar a cada usuario solo los accesos necesarios para su trabajo.
- Separación de funciones: Evitar que una sola persona tenga acceso a todas las funciones críticas (por ejemplo, desarrollo y producción).
- Acceso temporal: Para tareas específicas, otorgar permisos con límite de tiempo.

Manejo de accesos en proyectos o equipos

- Control centralizado: Usar sistemas de gestión de identidades, en la superintendencia de Sociedades se utiliza AzureDevops
- Registros de acceso (logs): Monitorear quién accede a qué, cuándo y desde dónde.

15.2. Manejo de Datos Sensibles

Estos lineamientos aplican para proteger datos en reposo (almacenados) y en tránsito (cuando se envían entre sistemas), tanto en el frontend como en el backend, almacenamiento de contraseñas (definir) y datos personales.

- Protección de datos en tránsito (red).
  - Siempre usar HTTPS (TLS 1.2 o superior)
  - Nunca enviar datos sensibles por HTTP.
  - Validar certificados SSL.
    - Aplicar HSTS (HTTP Strict Transport Security)
- Web APIs: JWT con HS256OAuth 2.1

15.3. Cabeceras en aplicaciones web

Estas cabeceras constituyen una defensa en profundidad (defense-in-depth) esencial para aplicaciones web modernas.

X-Frame-Options: SAMEORIGIN

Propósito: Impide que tu sitio sea cargado dentro de un <iframe> desde otro dominio (clickjacking).

Remediación: Protege contra ataques de clickjacking, donde un atacante puede engañar a los usuarios para que hagan clic en elementos de tu página que no ven.

Cache-Control: no-cache

Propósito: Les indica a los navegadores y proxies que no deben almacenar en caché la página. Debería evitar que datos sensibles se almacenen en la caché.

Remediación: Previene que información sensible o privada sea almacenada en cachés intermedias o del cliente, que podría ser accesible sin autenticación.

Connection: keep-alive

Propósito: Mantiene la conexión abierta entre el cliente y el servidor para múltiples solicitudes, evitando la sobrecarga de abrir y cerrar conexiones repetidamente.

Remediación: Aunque no directamente relacionado con la seguridad, mejora el rendimiento y reduce la carga del servidor.

X-Accel-Buffering: no

Propósito: Desactiva el almacenamiento en búfer de contenido en servidores de proxy (por ejemplo, Nginx).

Remediación: Evita que se almacenen en caché contenido sensible que podría no estar listo para su distribución inmediata. Es útil en entornos donde el proxy puede estar involucrado.

Content-Security-Policy (CSP)

Propósito: Define las fuentes de contenido permitidas en el sitio web, limitando el tipo de contenido que puede ser ejecutado por el navegador (scripts, estilos, imágenes, etc.).

Remediación: Protege contra XSS (Cross-Site Scripting) al limitar las fuentes de recursos externos que pueden ser ejecutados. En tu política:

default-src 'self': Solo permite cargar recursos desde el mismo origen.

script-src: Limita los scripts a dominios específicos, como Google y tu propio dominio.

frame-src: Permite solo ciertos orígenes en iframe, evitando que tu sitio sea embebido en sitios maliciosos (relacionado con la defensa contra clickjacking).

object-src 'none': Deshabilita el uso de object, embed y applet, lo que bloquea ataques como Flash-based XSS.

frame-ancestors 'self': Previene que tu sitio sea cargado como parte de otro sitio (otra defensa contra clickjacking).

En general, la política CSP refuerza la seguridad web y evita la ejecución de recursos maliciosos en tu página.

X-Content-Type-Options: nosniff

Propósito: Evita que el navegador adivine (sniff) el tipo de contenido de un archivo.

Remediación: Protege contra ciertos ataques de mimetype-sniffing donde el navegador podría intentar ejecutar un archivo con una extensión incorrecta (por ejemplo, un archivo JavaScript como si fuera texto plano o imagen).

X-XSS-Protection: 1; mode=block

Propósito: Activa el filtro de protección contra XSS en navegadores antiguos (como IE).

Remediación: Bloquea ataques XSS en el navegador al detectar patrones de inyección de scripts maliciosos. Aunque la mayoría de los navegadores modernos lo hacen de forma nativa, esta cabecera sigue siendo útil para compatibilidad.

Expect-CT: enforce, max-age=43200

Propósito: Habilita el Certificate Transparency para tu sitio, lo que obliga a los certificados SSL/TLS a ser registrados públicamente.

Remediación: Ayuda a prevenir ataques como la suplantación de certificados (donde un atacante podría emitir un certificado SSL falso). El max-age define el tiempo en segundos durante el cual el navegador debe aplicar esta política.

Feature-Policy: fullscreen 'none'

Propósito: Restringe el uso de ciertas APIs del navegador, en este caso deshabilita el uso de la función fullscreen en tu aplicación.

Remediación: Limita las capacidades del navegador y ayuda a reducir la superficie de ataque. En este caso, evita que tu sitio pueda ser puesto en pantalla completa, lo que podría ser una vía de ataque en algunos escenarios.

Permissions-Policy: fullscreen=()

Propósito: Similar a Feature-Policy, controla las funcionalidades de API que pueden ser utilizadas por el sitio. En este caso, restringe la funcionalidad de pantalla completa a ningún origen.

Remediación: Refuerza la seguridad al reducir las capacidades disponibles para el navegador y proteger contra abusos de características del navegador.

Referrer-Policy: no-referrer

Propósito: Impide que el navegador envíe la cabecera Referer al hacer solicitudes, lo que previene que sitios externos vean de dónde proviene el tráfico.

Remediación: Protege la privacidad del usuario al no exponer información sensible como URLs de referencia en las solicitudes HTTP.

strict-transport-security: max-age=31536000; includesubdomains

Propósito: Fuerza el uso de HTTPS para todas las conexiones en el dominio, asegurando que los usuarios nunca se conecten a través de HTTP.

Remediación: Previene ataques de downgrade (como man-in-the-middle), asegurando que todas las conexiones sean seguras. El max-age=31536000 especifica que esta política se debe aplicar durante 1 año, y includesubdomains incluye también los subdominios.

remove name="x-powered-by"

Propósito: Elimina la cabecera X-Powered-By que indica qué tecnología está utilizando el servidor (por ejemplo, PHP, ASP.NET).

Remediación: Previene que los atacantes puedan obtener información sobre la infraestructura del servidor, lo que podría ayudarles a identificar vulnerabilidades específicas del servidor o plataforma.

15.4. Uso de la Inteligencia artificial en el desarrollo

La IA no reemplaza los principios básicos de seguridad: Las sugerencias de la IA pueden introducir vulnerabilidades como inyecciones SQL, XSS o permisos incorrectos. Aplica siempre una revisión de seguridad manual, especialmente en código que maneje autenticación, autorización o datos sensibles.

15.4.1. Mentalidad Fundamental: "Engineer-in-the-Loop"

El principio más importante es que el desarrollador es el responsable final del código que se despliega. Las herramientas de IA, como GitHub Copilot o ChatGPT, son asistentes para aumentar la productividad, no para tomar decisiones autónomas. Tu rol es supervisar, validar y refinar su output.

15.4.2. Generar Código: Se debe especificar y controlar el proceso

Proporciona contexto detallado: En lugar de "escribe una función para conectarse a una BD", especifica el lenguaje, el framework, el tipo de base de datos y los requisitos de manejo de errores.

Divide tareas grandes: Se le debe pedir a la IA que genere código por componentes o funciones pequeñas. Esto facilita la revisión y reduce errores.

Seguir los estándares de este documento: configurar la herramienta para que alinee el código generado con las convenciones y guías de estilo de tu equipo.

Revisar código: no confiar al 100% lo que una herramienta de IA puede generar.

Revisa manualmente: Examinar el código generado línea por línea como si viniera de un compañero. Busca lógica errónea, posibles vulnerabilidades (como inyecciones SQL o XSS) y eficiencia.

Prueba rigurosamente: Ejecutar pruebas unitarias y de integración. La IA puede generar código que parece funcionar, pero falla en casos límite.

Verificar las dependencias: Si la IA sugiere usar una librería externa, se debe investigar su mantenimiento, seguridad y licencia antes de integrarla.

15.4.3. Seguridad de Datos y Código

Proteger la información sensible: Nunca se debe subir a una herramienta de IA pública código fuente completo con lógica de negocio, secretos (API keys, contraseñas), tokens, o datos personales de usuarios (PII). Se debe usar datos de prueba anonimizados.

Gestionar los secretos de forma segura: Usar siempre gestores de secretos (como Vault) o variables de entorno. Nunca se debe codificar en el código para luego ser enviado a la IA.

Defensas contra la inyección de prompts: Si la aplicación permite que los usuarios envíen prompts a un modelo de IA, se debe tratar sus entradas como no confiables; deben ser validadas y sanitizadas para evitar que manipulen el sistema y le hagan revelar información confidencial o ejecutar acciones no deseadas.

15.4.4. Manejo de Errores y Monitoreo

La IA puede generar código, comandos o "hechos" que son incorrectos, pero suenan plausibles. El desarrollador debe tener sano escepticismo y verifica con documentación oficial.

Registra (log) de forma segura: Para depurar, puede ser útil registrar las interacciones con la IA, pero el desarrollador debe asegurarse de que estos logs no capturen información confidencial. Se debe implementar monitoreo para detectar patrones de uso anómalos que puedan indicar un ataque.

15.5. Consideraciones de Propiedad Intelectual (Derechos de autor)

En la superintendencia de sociedades se debe considerar el uso de la IA bajo la siguiente política.

15.5.1. Propósito y Alcance

El presente documento establece los lineamientos, restricciones y buenas prácticas para el uso de herramientas de Inteligencia Artificial Generativa (IA) en las actividades de diseño, desarrollo y mantenimiento de software de la Superintendencia de Sociedades. Su objetivo es potenciar la productividad y la innovación del equipo de tecnología, al mismo tiempo que se mitigan riesgos críticos relacionados con la seguridad de la información, la protección de la propiedad intelectual institucional y el cumplimiento normativo.

Esta política aplica a todo el personal y contratistas que, en el marco de sus funciones, utilicen herramientas de IA (como ChatGPT, GitHub Copilot, Claude, o similares) para generar, analizar o modificar código fuente, documentación técnica, diseños de arquitectura o cualquier otro activo relacionado con los sistemas de información de la entidad.

15.5.2. Principios Fundamentales

El uso de IA en la Superintendencia de Sociedades se rige por los siguientes principios irrenunciables:

i. **Confidencialidad y Soberanía de la Información:** El código fuente, los modelos de datos, las configuraciones y cualquier activo digital desarrollado por o para la entidad son propiedad intelectual estratégica y están protegidos por derechos de autor. Su confidencialidad es prioritaria.
ii. **Seguridad como Fundamento:** Todas las interacciones con herramientas de IA deben realizarse bajo un paradigma de seguridad por diseño, previniendo fugas de información y vulnerabilidades.
iii. **Responsabilidad Humana y Supervisión:** La IA es un instrumento de apoyo. El desarrollador o analista que la utiliza es el único y último responsable de la calidad, seguridad, corrección y pertinencia del trabajo final entregado.
iv. **Cumplimiento y Autorización:** Su uso en proyectos específicos requiere una evaluación y autorización previa, en línea con los procedimientos establecidos en esta política.

15.5.3. Prohibiciones Explícitas

Se prohíbe terminantemente, so pena de incurrir en faltas disciplinarias graves:

- Subir, copiar, pegar o ingresar en cualquier herramienta de IA de carácter público (cloud), total o parcialmente:
  - Código fuente completo de proyectos o módulos.
  - Fragmentos de código que contengan lógica de negocio propia, algoritmos sensibles o estructuras de datos específicas de la Superintendencia.
  - Archivos de configuración de sistemas (ej., application.properties, web.config).
  - Queries de bases de datos con estructuras reales.
  - Cualquier secreto: claves API, contraseñas, certificados digitales, tokens, o variables de entorno.
  - Documentación interna que detalle el funcionamiento de sistemas críticos.
- Utilizar herramientas de IA no evaluadas y aprobadas por la Dirección de Tecnología.
- Implementar en los sistemas de producción código generado por IA que no haya pasado por el proceso completo de revisión de pares (code review) y pruebas unitarias y de integración.

15.5.4. Usos Permitidos y Buenas Prácticas

El uso de la IA está autorizado como herramienta de apoyo en las siguientes actividades, cumpliendo siempre las restricciones de la sección 3:

Diseño y Arquitectura:

- Generar explicaciones o analogías para conceptos arquitectónicos complejos.
- Brainstorming de nombres para componentes, variables o clases siguiendo convenciones (sin revelar lógica).
- Revisar y sugerir mejoras a documentos de diseño genéricos (ej., diagramas de secuencia despersonalizados).

Desarrollo y Codificación:

- **Generación de código "verde" o nuevo**: Crear componentes, funciones o clases nuevas que no existan en el código base actual, utilizando como contexto solo descripciones funcionales en lenguaje natural.
  - Ejemplo Permitido: "Genera una función en Java que valide el formato de un RUT Colombiano, sin usar librerías externas."
  - Ejemplo Prohibido: Copiar y pegar la función de validación de RUT existente en nuestro sistema y pedirle a la IA que la "optimice".
- **Explicación de código público:** Pedir explicaciones de errores, funciones de librerías de código abierto o estándares públicos (APIs de Java, .NET, React).
- **Generación de pruebas unitarias:** Crear esquemas de pruebas para nuevas funcionalidades, basándose en la especificación, no en el código implementado.
- **Refactorización segura:** Solicitar sugerencias para refactorizar código genérico (ej., patrones de bucles, manejo de excepciones estándar), siempre que el código específico de negocio no sea expuesto.

Mantenimiento y Documentación

- Generar comentarios en el código a partir de descripciones funcionales.
- Ayuda para redactar documentación técnica de alto nivel.

- Traducción de documentación técnica entre idiomas.

15.5.5. Procedimiento de Autorización y Uso Seguro

Evaluación y Solicitud: El líder técnico del proyecto evaluará la necesidad de usar IA para tareas específicas. De considerarlo necesario, completará el Formulario de Solicitud de Uso de IA (Anexo A), detallando la herramienta a usar, el propósito y las medidas de seguridad para evitar fugas.
Autorización: La solicitud debe ser aprobada por escrito por el director de Tecnología o su coordinador. Sin esta autorización, no se permite el uso de herramientas de IA en el proyecto.
Configuración Segura: Para herramientas configurables (ej., GitHub Copilot Business), se habilitará el modo privado o empresarial, que garantiza que los prompts y el código no se utilicen para entrenar modelos públicos.
Capacitación Obligatoria: Todo personal autorizado debe completar el curso de "**Uso Seguro de IA para Desarrollo**" impartido por la dirección de Tecnología de la Información y las Comunicaciones.
Registro y Auditoría: Se deberá registrar en la bitácora del proyecto el uso de IA para tareas relevantes. La Dirección de Tecnología y la Oficina de Seguridad de la Información realizarán auditorías periódicas para verificar el cumplimiento.

15.5.6. Responsabilidades

Director de Tecnología: Autorizar el uso, establecer la política y supervisar su cumplimiento.
Líder técnico y de proyecto: Garantizar que su equipo conoce y aplica la política, y solicitar las autorizaciones necesarias.
Desarrolladores y analistas: Utilizar las herramientas de forma responsable, ceñirse estrictamente a los usos permitidos y ser responsables últimos del código que producen o modifican.
Oficina de Seguridad de la Información: Brindar capacitación, realizar auditorías y monitorear posibles incidentes de fuga de información.

15.5.7. Infracciones y Sanciones

El incumplimiento de esta política se considerará una falta al régimen disciplinario de la entidad y a los deberes de confidencialidad, pudiendo acarrear sanciones que incluyen, pero no se limitan a: amonestación escrita, suspensión, terminación del contrato de trabajo o de prestación de servicios, y las acciones legales correspondientes por violación a la propiedad intelectual y daños a la entidad.

15.6. Revisión de Seguridad

- En la superintendencia el proceso de seguridad antes del despliegue consiste en realizar una revisión de los aspectos relacionados con la protección de la aplicación y su entorno antes de ponerla en producción. Esto incluye verificar que la autenticación y autorización estén correctamente implementadas, que no existan credenciales expuestas en el código ya que se configuran en archivos paramétricos denominados webconfig, y que se apliquen buenas prácticas como el principio de mínimo privilegio. También se validan las configuraciones de seguridad en la infraestructura, se asegura el cifrado de datos y comunicaciones, se revisan las cabeceras de seguridad, y se ejecutan pruebas para detectar vulnerabilidades conocidas en el código y las dependencias se realiza la revisión de la corrección de los bugs o tareas asignadas previamente en el sprint planing Finalmente, se confirman aspectos operativos como la gestión de logs, backups, monitoreo activo Análisis de vulnerabilidades

- El análisis de código estático es un proceso que consiste en revisar el código fuente de una aplicación sin ejecutarlo, con el objetivo de identificar errores, vulnerabilidades de seguridad, malas prácticas de programación y posibles incumplimientos de estándares de codificación. Esta técnica se realiza dentro de la superintendencia con la herramienta automatizada que realiza el escaneo del código línea por línea para detectar patrones sospechosos o riesgos potenciales, como inyecciones, uso inseguro de variables, o lógica defectuosa. El análisis de código estático es especialmente útil en etapas tempranas del desarrollo, ya que permite corregir problemas antes del despliegue, reduciendo así los

costos y el impacto de los errores en producción. Además, fomenta la calidad del software y la seguridad como parte integral del ciclo de desarrollo.

16. CONTROL DE CAMBIOS

Versión	Fecha	Descripción del Cambio
001	11/6/2025	Creación del documento
002	11/09/2026	Se actualizan y agregan lineamientos que se tendrán en cuenta a la hora de desarrollar software en la Entidad.

Elaboró	Revisó	Aprobó
Nombre: Marisol Castiblanco	Nombre: Marisol Castiblanco Cargo: Coordinador Grupo de Sistemas y Arquitectura –	Nombre: Ricardo Fernelix Ríos

Cargo: Profesional Grupo de Sistemas y Arquitectura – Profesional Grupo de Innovación, Desarrollo y Arquitectura de Aplicaciones Fecha: 31 de agosto de 2026	Coordinador Grupo de Innovación, Desarrollo y Arquitectura de Aplicaciones Fecha: 31 de agosto de 2026	Cargo: director de Tecnologías de la Información y las Comunicaciones Fecha: 11 de septiembre de 2026