PERÚ | Ministerio de Educación

“Directiva para el uso de Marcos de Trabajo, Herramientas y Buenas Prácticas para el desarrollo estandarizado de productos de *software* en el MINEDU”

FIRMA DIGITAL

Firmado digitalmente por: VILCHEZ INGA Cesar FAU 20131370998 hard
Motivo: En señal de conformidad
Fecha: 15/08/2022 15:55:07-0500

Resolución de Aprobación			
Código	Versión	Páginas	Fecha Aprobación
	1.0	90	

DIRECTIVA PARA EL USO DE MARCOS DE TRABAJO, HERRAMIENTAS Y BUENAS PRÁCTICAS PARA EL DESARROLLO ESTANDARIZADO DE PRODUCTOS DE *SOFTWARE* EN EL MINEDU

1. OBJETIVO

Establecer las disposiciones técnicas que permitan el desarrollo de productos de *software* bajo estándares de calidad y seguridad, en concordancia con los dispositivos legales vigentes y las buenas prácticas, cuyos fundamentos son reconocidos y aplicables a nivel nacional e internacional.

2. FINALIDAD

Definir un conjunto de disposiciones técnicas para el proceso de desarrollo de productos de *software* para el MINEDU, teniendo en cuenta la necesidad de obtener productos de calidad, con la aplicación de métodos, técnicas y prácticas que aseguren la integración, interoperabilidad, escalabilidad, bajo acoplamiento de los sistemas desarrollados y garantizando la integridad, disponibilidad y confidencialidad de la información.

3. ALCANCE

Las disposiciones contenidas en la presente Directiva son aplicables a todos los órganos y unidades orgánicas del MINEDU que directa o indirectamente participen en el proceso de desarrollo de productos de *software* que posteriormente requieran ser alojados en los ambientes de producción de la plataforma tecnológica del MINEDU.

4. BASE NORMATIVA

4.1. Ley N° 27658, Ley Marco de Modernización de la Gestión del Estado.
4.2. Ley N° 29733, Ley de Protección de Datos Personales.
4.3. Decreto Legislativo N° 1412, que aprueba la Ley de Gobierno Digital.
4.4. Decreto de Urgencia N° 007-2020, que aprueba el marco de confianza digital y dispones medidas para su fortalecimiento.
4.5. Decreto de Urgencia N° 006-2020, que crea el Sistema Nacional de Transformación Digital.
4.6. Decreto Supremo N° 004-2013-PCM, que aprueba la Política Nacional de Modernización de la Gestión Pública.
4.7. Decreto Supremo N° 003-2013-JUS, que aprueba el Reglamento de la Ley N° 29733, Ley de Protección de Datos Personales.
4.8. Decreto Supremo N° 001-2015-MINEDU, que aprueba el Reglamento de Organización y Funciones del Ministerio de Educación.
4.9. Decreto Supremo N° 029-2021-PCM, que aprueba el Reglamento del Decreto Legislativo N° 1412, Decreto Legislativo que aprueba la Ley de Gobierno Digital, y establece disposiciones sobre las condiciones, requisitos y uso de las tecnologías y medios electrónicos en el procedimiento administrativo.
4.10. Decreto Supremo N° 157-2021-PCM, que aprueba el Reglamento del Decreto de Urgencia N° 006-2020, Decreto de Urgencia que crea el Sistema Nacional de Transformación Digital.

4.11. Resolución Ministerial N° 004-2016-PCM, que aprueba el uso obligatorio de la Norma Técnica Peruana "NTP-ISO/IEC 27001:2014 - Tecnología de la Información. Técnicas de Seguridad. Sistemas de Gestión de Seguridad de la información. Requisitos. 2ª Edición", en todas las entidades integrantes del Sistema Nacional de Informática.

4.12. Resolución Ministerial N° 041-2017-PCM, que aprueba el uso obligatorio de la Norma Técnica Peruana "NTP ISO/IEC 12207: 2016 - Ingeniería de Software y Sistemas. Procesos del ciclo de vida del software. 3a Edición", en todas las entidades integrantes del Sistema Nacional de Informática.

4.13. Resolución de Secretaría de Gestión Pública N° 006-2019-PCM/SGP, que aprueba la Norma Técnica Nº 001-2019-PCM-SGP, Norma Técnica para la Gestión de la Calidad de Servicios en el Sector Público.

4.14. Resolución de Secretaría General N° 028-2019-MINEDU, que aprueba la Directiva Nº 003-2019-MINEDU/SPE-OTIC, denominada "Verificación y Validación de producto software de desarrollo externo".

4.15. Resolución de Secretaría General N° 140-2021-MINEDU, que aprueba el procedimiento denominado "Gestionar los Riesgos organizacionales".

4.16. Resolución de Secretaría General N° 148-2021-MINEDU, que aprueba la Directiva para la Gestión de la Seguridad de la información del Ministerio de Educación.

4.17. Resolución Directoral N° 019-2013-JUS/DGPDP, que aprueba la Directiva de Seguridad de la Información Administrada por los Bancos de Datos Personales.

5. GLOSARIO DE TÉRMINOS y SIGLAS

5.1. DEFINICIONES

- **Ambiente de certificación:** Infraestructura tecnológica donde se despliega el producto de *software* para la realización de las pruebas de control de calidad y seguridad. Los ambientes de certificación son administrados por el equipo responsable de implementar y administrar la infraestructura tecnológica.
- **Ambiente de producción:** Infraestructura tecnológica donde se despliega el producto de *software* y en versión utilizable para los usuarios finales. Los ambientes de producción son administrados por el equipo responsable de implementar y administrar la infraestructura tecnológica.
- **Apetito del riesgo:** Expresa los niveles de riesgos dispuestos a ser aceptados conforme a la clasificación de la información involucrada en el riesgo y al logro de los objetivos que se quieren alcanzar como negocio.
- ***API*:** *Application Programming Interface* (en español Interfaz de Programación de Aplicaciones). Es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el *software* de las aplicaciones. Estas permiten que los productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados, simplificando de esta manera el desarrollo de las aplicaciones.
- ***API Gateway*:** En español Puerta de Enlace. Es un componente de *software* encargado de unificar o desacoplar la interfaz que "ven" los clientes (en este caso, los consumidores de las *API* que podrían ser aplicaciones móviles o *web*) de la implementación de los microservicios. Funciona como un *proxy* inverso, con roles para autenticación y el control de acceso a microservicios. También puede asumir el rol de balanceador de carga.

- **Arquitectura digital:** Es una disciplina aplicada a la búsqueda de soluciones en el ámbito de las tecnologías de la información y comunicaciones aplicando técnicas y metodologías de diseño.
- ***DevSecOps*:** Proviene del acrónimo DEV (Desarrollo), SEC (Seguridad) y OPS (operaciones). Es un enfoque para la entrega de *software* que tiene como ventaja la integración de la seguridad desde el inicio del desarrollo y, por lo tanto, promueve un enfoque de seguridad continua, por lo cual los equipos de desarrollo y operaciones colaboran para crear, probar, implementar y monitorear aplicaciones con velocidad, seguridad, calidad y control.
- **Equipo de desarrollo**: Equipo multidisciplinario encargado del análisis, diseño, construcción y aseguramiento de calidad de un producto o componente de *software*. Este equipo cuenta con al menos un integrante con capacidades para identificar vulnerabilidades de seguridad del producto de software, resuelve y comparte la solución implementada.
- **GIT**: Es un *software* de control de versiones de aplicaciones. Su propósito es llevar el registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.
- ***Gitflow***: Es un modelo alternativo de creación de ramas en *GIT* en el que se utilizan ramas de función y varias ramas principales y de esta forma, dar más fluidez al proceso.
- ***Github Flow***: Es una alternativa simple y ligera a *GitFlow*. Se basa en un flujo de trabajo basado en ramas que permite a equipos de desarrollo enfocarse principalmente en la entrega continua y está pensado para que la implementación en producción ocurra con frecuencia, incluso varias veces al día si es posible.
- ***GitLab Flow***: Es una alternativa más simple a *GitFlow* y combina desarrollo basado en funciones y ramas de funciones con seguimiento de problemas. Incluye un conjunto de mejores prácticas y pautas para garantizar que los equipos de desarrollo de software sigan un proceso fluido para enviar funciones de forma colaborativa.
- **Herramienta (tecnológica):** Es cualquier "*software*" o "*hardware*" que ayuda a realizar bien una tarea, entendiéndose por "realizar bien" que se obtengan los resultados esperados, como ahorro de tiempo y ahorro en recursos personales y económicos.
- ***HTTP*:** *Hypertext Transfer Protocol* (en español Protocolo de Transferencia de Hipertexto). Es un protocolo de comunicación de *Internet* que permite las transferencias de información a través de archivos en la *Web* y que funciona por peticiones y respuestas. Es un protocolo que no está basado en el estado, lo cual significa que el protocolo trata cada petición como una transacción independiente y no tiene en cuenta cualquier solicitud anterior.
- **IDE:** *Interface Development Environment* (En español Entorno de Desarrollo Integrado) es un sistema de software para el diseño de aplicaciones que combina herramientas comunes para desarrolladores en una sola interfaz de usuario gráfica (dinámica).
- **JSON:** *JavaScript Object Notation* (En español Notación de Objetos de JavaScript). Es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato independiente del lenguaje.

- ***LTS*: *Long Term Support*** (en español Soporte a Largo Plazo). Es un término utilizado para clasificar versiones o ediciones especiales de *software* para tener soporte durante un período más amplio que el normal.
- ***MVP*: *Minimum viable product*** (en español Producto Mínimo Viable). Es una versión de un producto con funciones mínimas para recabar la mayor cantidad de aprendizaje y/o proporcionar retroalimentación para los desarrollos futuros. Es usado para probar rápidamente de manera cuantitativa y cualitativa la respuesta de los usuarios finales frente a un producto y/o una funcionalidad.
- **Marco de trabajo:** Es un conjunto de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.
- **Microservicios:** Enfoque para desarrollar una aplicación como un conjunto de pequeños servicios, cada uno encargado de la implementación de una funcionalidad específica y ejecutándose en su propio proceso y comunicándose con mecanismos ligeros, a menudo una *API* a través de *HTTP/HTTPS*.
- ***Pentesting*:** Es una abreviatura formada por dos palabras "*penetration*" y "*testing*" y es una práctica/técnica que consiste en atacar diferentes entornos o sistemas con la finalidad de encontrar vulnerabilidades y prevenir posibles fallos de seguridad en el mismo.
- **Pruebas Unitarias:** Son una forma de comprobar el correcto funcionamiento de una unidad de código. Además de verificar que el código hace lo que tiene que hacer, se verifica que sean correctos los nombres, tipos de los parámetros, las rutas, los datos retornados entre otros.
- **REST:** Es cualquier interfaz entre sistemas que use *HTTP* para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como *XML y JSON.*
- ***SQL*:** *Structured Query Language* (en español Lenguaje de Consulta Estructurado). Es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas.
- ***TO-BE:*** Es parte de un mapeo de procesos, está estipulando a dónde quieres llegar al final de la evolución del proceso. El mapa debe estar alineado con la planificación estratégica de la organización en su conjunto, para que los objetivos se logren de manera más eficiente. En esta segunda fase, se rediseña el proceso, utilizando el Modelado y Notación de Procesos de Negocio (BPMN).
- ***XLM:*** *Extensible Markup Languaje* (en español Lenguaje de Marcas Extensible), es un formato de texto que se utiliza para almacenar e intercambiar datos estructurados, bien sea que se trate de documentos, configuraciones, transacciones o simplemente datos. Es un lenguaje de marcado que define la estructura y el significado de los datos.
- ***UML:*** *Unified Modeling Language* (en español Lenguaje Unificado de Modelado) fue creado para forjar un lenguaje de modelado visual común y semántica y sintácticamente rico para la arquitectura, el diseño y la implementación de sistemas de software complejos, tanto en estructura como en comportamiento.

5.2. SIGLAS

- **MINEDU:** Ministerio de Educación.
- **OTIC:** Oficina de Tecnologías de la Información y Comunicación.

6. DISPOSICIONES GENERALES

La implementación progresiva de una arquitectura digital[1] para el MINEDU que permite aunar esfuerzos en la modernización de la gestión del sistema educativo requiere lo siguiente:

6.1. Se haga uso de un conjunto de herramientas, patrones, metodologías marcos de trabajo, buenas prácticas, entre otros, que permitan orientar los esfuerzos del desarrollo de *software* eficiente y seguro con un enfoque ágil, colaborativo, con miras a conformar un ecosistema al interior de MINEDU y que se integre a diversas soluciones de otros sectores del Estado.

6.2. Se desarrolle productos de *software* y/o sistemas distribuidos orientados a atender la gestión educativa, que sean de calidad y altamente escalables mediante modernas herramientas y técnicas en lo referente a base de datos, que se encuentren alineados con los principios de alta cohesión, bajo acoplamiento de los componentes de los sistemas.

6.3. Se promueva el desarrollo de productos de *software* que tengan la facultad de proveer información en: a) Otros sistemas de información de nuestra institución, b) Sistemas de información y/o aplicaciones de otras instituciones y c) al público en general; sin la necesidad de complejos y costosos mantenimientos.

6.4. Se promueva el desarrollo estandarizado, que permita establecer un medio de comunicación e integración entre los diversos productos y/o servicios del MINEDU, así como facilitar la interoperabilidad con instituciones externas.

6.5. Se adopte un enfoque que permita integrar la seguridad en los procesos de desarrollo y operaciones de los productos de *software* que serán puestos a disposición de la comunidad educativa, contribuyendo en el cumplimiento del enfoque de seguridad: *DEVSECOPS*.

6.6. Se utilice las interfaces de programación de aplicaciones (en adelante las *API*) como mecanismo para poder interactuar con fuentes de información, tanto internas o externas, de tal forma que los recursos, productos o servicios se comuniquen con otros, sin necesidad de saber cómo están implementados, incorporando flexibilidad y escalabilidad, permitiendo ahorrar esfuerzo en el desarrollo de sistemas y expandir fácilmente sus funcionalidades. Este escenario exige que la creación y documentación de las *API* se realicen con buenas prácticas que contribuyan al éxito de las implementaciones y la satisfacción de los usuarios finales.

6.7. En el desarrollo de nuevos productos de *software*, se adopte principalmente el uso de microservicios y orquestadores de servicios, sobre el uso de arquitecturas monolíticas.

7. DISPOSICIONES ESPECÍFICAS

7.1. CONSIDERACIONES PARA EL DESARROLLO DE UN *SOFTWARE* DE CALIDAD EN EL MINEDU

7.1.1. Todo desarrollo de un producto de *software*, se inicia con la identificación de las necesidades del área usuaria.

7.1.2. Para la definición de los requerimientos funcionales se tiene en cuenta al menos las siguientes consideraciones:

a. Definir una o más características del *software* a desarrollar.

[1] En el Anexo Nº.01 se muestra la Propuesta de Arquitectura Digital desde la perspectiva del Sector Educativo.

b. Ser claros, completos y concretos, evitando ambigüedades.
c. Permitir identificar su propósito y los criterios de aceptación, para verificar su cumplimiento.
d. Permitir la priorización, para distinguir si son obligatorios, deseables u opcionales.
e. No deben contener diversos requerimientos (deben ser específicos).

7.1.3. Se coordinan mesas de trabajo con el área usuaria para que, a través de ellas, se analice con mayor detalle los requerimientos, se defina la oportunidad y priorización de la lista de requerimientos.

7.1.4. De acuerdo a la naturaleza del proyecto se elabora las especificaciones de casos de uso o las historias de usuario, tomando en cuenta inicialmente las funcionalidades que constituyen el producto mínimo viable, de existir.

7.1.5. Se identifica los requerimientos no funcionales, así como también los requerimientos similares, que hayan sido atendidos previamente, ya que pueden formar parte de un proceso de reutilización.

7.1.6. Se establece el apetito de riesgo acotado al proyecto y los requisitos de seguridad de información, en concordancia con la normativa institucional vigente y a la naturaleza del proyecto.

7.1.7. Se elaboran los documentos de especificación de requerimientos alineados a las necesidades de los usuarios.

7.1.8. Se selecciona el tipo de base de datos.

7.1.9. Se elaboran los documentos de arquitectura del sistema, el diseño o rediseño detallado de los componentes de *software*.

7.1.10. Se seleccionan las herramientas tecnológicas de desarrollo, de calidad y seguridad de *software* entre otros.

7.1.11. Con la arquitectura inicial del sistema de información y los requerimientos no funcionales identificados, se proyecta la arquitectura de infraestructura tecnológica del sistema o solución.

7.1.12. El diseño de componentes de *software* se encuentra alineado a:

a. Las necesidades de los usuarios.
b. A los documentos normativos vigentes y
c. A las buenas prácticas de la industria tecnológica.

7.1.13. Para la elaboración de los documentos técnicos, se seleccionan los formatos existentes en el repositorio de documentos.

7.1.14. Se realiza el control de versiones de los artefactos.

7.1.15. Los componentes y la base de datos se diseñan de acuerdo a las especificaciones y/o historias de usuario, tomando en cuenta las reglas de negocio y/o restricciones identificadas considerando:

a. Hacer uso adecuado del repositorio de código fuente.
b. Aplicación de los principios y patrones de diseño.
c. Cumplimiento del diseño de la arquitectura propuesta y en los (re)diseños de componentes.

7.1.16. Conforme se va desarrollando cada componente se ejecuta de forma paralela las pruebas unitarias, para asegurar la calidad de cada producto.

7.1.17. Se elabora y/o actualiza el manual de instalación, configuración y despliegue.

7.1.18. Se instala y configura el ambiente de desarrollo de acuerdo con las características y los requisitos de *hardware* y *software* descritos en el manual de instalación, configuración y despliegue.

7.1.19. De acuerdo a la programación de integraciones, los componentes desarrollados y probados unitariamente son desplegados en los ambientes de desarrollo.

7.1.20. En el ambiente de desarrollo, las integraciones y los componentes desarrollados son probados de forma integral.
7.1.21. Se verifica la elaboración de los documentos técnicos consignados en la lista de verificación del documento de pase a producción.
7.1.22. Se despliegan las integraciones, los componentes desarrollados y probados en el ambiente de certificación.
7.1.23. Se define las métricas de calidad aceptables en el MINEDU, bajo los criterios de razonabilidad adecuados a las necesidades de cada proyecto.
7.1.24. Se realizan las pruebas de calidad y seguridad, a fin de obtener una versión funcional estable que cumpla con las reglas y restricciones del negocio.
7.1.25. De acuerdo a las características y los requisitos de *hardware* y *software* descritos en el manual de instalación, configuración y despliegue, se despliega en el ambiente de producción, las integraciones, los componentes desarrollados y probados llamados producto de *software*. Este documento varía dependiendo del tipo de despliegue que se realice.
7.1.26. Se administra y se pone a disposición, una base de conocimiento que contiene las soluciones a las vulnerabilidades de seguridad reportadas en el *pentesting* aplicado a los productos de *software* evaluados. Esta base de conocimiento es actualizada por los especialistas involucrados en el desarrollo y pruebas del producto de *software*.

7.2. HERRAMIENTAS TECNOLÓGICAS PARA LA IMPLEMENTACIÓN ESTANDARIZADA DE PRODUCTOS DE *SOFTWARE* PARA EL MINEDU

7.2.1. SELECCIÓN Y USO DE LAS HERRAMIENTAS TECNOLÓGICAS, PATRONES Y MARCOS DE TRABAJO EN EL DESARROLLO DE *SOFTWARE*

a. El equipo responsable del diseño de la arquitectura de *software* o quien cumpla su rol brinda orientación a los líderes técnicos de los equipos de desarrollo en la selección de Marcos de Trabajo, Patrones, Principios y Herramientas Tecnológicas, aprobadas por la OTIC y descritas en el **Anexo Nº 02 de la presente Directiva**, cuyo proceso de actualización estará a cargo de la OTIC sujeto a una evaluación técnica.
b. El equipo de desarrollo selecciona un marco de trabajo, de acuerdo al enfoque de ciclo de vida de gestión del proyecto que se haya determinado.
c. El equipo de desarrollo investiga y aprende permanentemente sobre el uso adecuado de las herramientas seleccionadas para el desarrollo del producto de *software*.
d. El equipo de desarrollo se entrena en la aplicación de las diferentes soluciones basadas en algoritmos, patrones, modelos, principios y arquitecturas.

7.2.2. REQUISITOS DE LAS HERRAMIENTAS TECNOLÓGICAS A SER UTILIZADAS EN EL MINEDU

Para la implementación de un enfoque *DEVSECOPS* en el MINEDU se debe seleccionar herramientas tecnológicas que permitan obtener mayor eficiencia, productividad y agilidad en el proceso integrado de desarrollo, pruebas y puesta en producción de los productos de *software*. Las herramientas tecnológicas a utilizarse en el MINEDU deben cumplir los siguientes requisitos:

a. Herramientas *open source* bajo la modalidad de licenciamiento comunitario.
b. Herramientas que se ejecutan en *on-premise* y en nube (privada y/o pública).
c. En caso de ser herramientas propietarias, MINEDU debe contar con licencias o suscripciones, de preferencia a perpetuidad.
d. Herramientas que, a menos que se indique lo contrario, deben ser utilizadas en su versión *LTS* o en su defecto la última versión.
e. Para el caso de desarrollo de *software* en nube, se podrá hacer uso de herramientas propietarias de cada proveedor.

7.2.3. GESTIÓN DEL REPOSITORIO

a. Con la finalidad de gestionar un único repositorio de código fuente, el MINEDU ha establecido a *GIT*, como la herramienta para administrar el código fuente, al cual sólo se accederá desde la red local de MINEDU y a través de la red informática corporativa.
b. El equipo responsable del diseño de la arquitectura de *software* o quien cumpla ese rol brinda orientación al líder técnico y/o quien haga sus veces en el equipo de desarrollo sobre acceso al repositorio.
c. El equipo responsable del diseño de la arquitectura de *software* o quien cumpla ese rol coordina la ejecución de las tareas de respaldo, las que están bajo responsabilidad del equipo responsable de los servidores y almacenamiento.
d. El líder técnico y/o quien haga sus veces en el equipo de desarrollo brinda al equipo de desarrollo el acceso al repositorio.
e. El equipo de desarrollo resguarda el incremento de código en la herramienta de versionamiento.
f. Para poder acceder a la base de conocimiento, que contiene los lineamientos para el uso de la herramienta de versionamiento y de la *IDE* utilizada en el MINEDU, los integrantes de los equipos de desarrollo cuentan con las credenciales de acceso a la red de MINEDU y conforman el grupo de desarrollo.
g. Los flujos de trabajo en el MINEDU deben estar basados en *Gitflow, Github Flow* y *Gitlab Flow*, tomando las mejores prácticas de ellas y adaptándolas al desarrollo propio.

7.2.4. VERSIONADO DE CÓDIGO FUENTE

El equipo de desarrollo debe tomar en cuenta las consideraciones técnicas en el uso de los flujos de trabajo descritas en el **Anexo Nº 03 de la presente Directiva.**

7.3. DISEÑO ESTANDARIZADO DE LAS BASES DE DATOS QUE UTILIZA EL MINEDU

7.3.1. SELECCIÓN Y USO DE LA BASE DE DATOS

a. El equipo responsable del diseño de la arquitectura de *software* o quien cumpla su rol identifica la o las bases de datos a ser utilizadas en el desarrollo.

b. El equipo responsable del diseño de la arquitectura de *software* o quien cumpla su rol realiza un análisis basado en los requerimientos funcionales y no funcionales, teniendo en consideración las características y diferencias entre los tipos de base de datos. En el caso que se trate de un sistema distribuido aplica el análisis del Teorema de CAP. Este análisis y sus conclusiones son incluidos en el formato vigente del documento "Arquitectura de *software*". Para el análisis, consultar en el **Anexo Nº 04 de la presente. Directiva**:
   - Teorema de CAP para Sistemas Distribuidos.
   - Diferencias entre el tipo de Base de Datos.

c. Se encuentran autorizadas los siguientes tipos de bases de datos en el desarrollo de aplicaciones[2]

Tabla N°1: Tipo de Base de Datos

TIPO	PRODUCTOS A SER UTILIZADOS
SQL	SQL Server, MySQL y PostgreSQL
NoSQL	MongoDB.

Fuente: Elaboración propia

d. El equipo responsable del diseño de la arquitectura de *software* o quien cumpla su rol en conjunto con el responsable de la administración de la base de datos selecciona el tipo o tipos de base de datos a utilizar en el desarrollo de *software*.

e. El equipo responsable del diseño de la arquitectura de *software* o quien cumpla su rol establece las restricciones a tener en cuenta en la definición de la ubicación física[3] de la base de datos (*on-premise* o alojado en la nube).

7.3.2. DISEÑO Y DOCUMENTACIÓN DE UNA BASE DE DATOS

El equipo de desarrollo debe tener en cuenta lo siguiente:

a. El diseño de la base de datos debe ser plasmado en un modelo o conjunto de modelos.
b. Los modelos conceptuales se elaboran separando los archivos por dominios y base de datos de forma independiente al tipo de base de datos seleccionado.
c. Utilizar el formato vigente del documento "Diseño Detallado" para incluir los detalles del diseño de bases de datos.
d. Utilizar diferentes patrones de diseño asociados al diseño de base de datos.
e. Utilizar una herramienta de modelamiento, con la debida licencia y/o suscripción, si esta es propietaria.
f. De considerar que el modelamiento es un proceso iterativo e incremental, donde se pueden adoptar dos enfoques, dependiendo de la estrategia adoptada para el desarrollo del producto de *software*:
   - Por producto mínimo viable: Se debe generar la o las bases de datos que soportan el *MVP*: modelo conceptual y modelo físico.

[2] Existen otros productos de bases de datos que son utilizados en MINEDU, pero no son considerados en este listado debido a que su presencia responde al uso de software prefabricado, por ejemplo, los siguientes productos: *Kong, Keycloak,* entre otros.
[3] Las consideraciones del presente documento son independientes de la ubicación física de la base de datos.

- Por entregas modulares parciales: Se debe generar un modelo conceptual de todo el sistema y para cada módulo a ser entregado, se requieren los modelos físicos respectivos.

g. Utilizar los siguientes insumos para diseñar el modelo de base de datos correcto:
   - Modelos de procesos *TO-BE*: Todas las bases de datos indicadas en los modelos de procesos *TO-BE* deben estar contenidas como bases de datos en el modelo conceptual.
   - Diagrama de arquitectura: Todas las bases de datos indicadas en el diagrama de arquitectura propuesta deben estar contenidas en el modelo conceptual de la base de datos.
   - Otros diseños de base de datos: El equipo responsable del diseño de la arquitectura de *software* o quienes cumplan su rol, podrán proveer de diseños elaborados previamente.

   Es posible tomar como referencia modelos aplicados en realidades similares externas[4].
h. Para el modelamiento de bases de datos relacionales tener en cuenta la siguiente secuencia de actividades y consideraciones:
   - Obtener el modelo lógico y modelo físico a partir del modelo conceptual.
   - Se recomienda que cada diagrama de base de datos no debe contener más de 20 entidades.
   - Los *scripts* de creación de tablas y procedimientos pueden ser obtenidos a partir del correspondiente modelo físico.
   - La documentación para el Diccionario de datos debe ser obtenida desde la herramienta de reportes de los modelos físicos.
   - Las estimaciones en volumen de almacenamiento deben ser obtenidas a través de la herramienta que se encuentran para tal fin en los modelos físicos de la herramienta.
   - Se debe optimizar el uso de los tipos de datos, de tal forma que se evite el exceso de espacio de almacenamiento.

i. Para las bases de datos *NoSQL*: En la elaboración de modelos físicos será preferible modelar con diagramas de clase ya que el modelo obtenido contiene los esquemas básicos de los documentos.
j. La revisión de los modelos conceptuales se debe realizar luego de una o varias reuniones de presentación y sustento solicitada por el gestor o quien haga sus veces o por el líder técnico o el responsable a cargo del desarrollo del sistema.
k. Presentar los modelos conceptuales de las aplicaciones y servicios al equipo responsable del diseño de la arquitectura de *software* para la aprobación correspondiente y posterior administración, resguardo y distribución.

7.3.3. USO Y MODIFICACIÓN DE LAS BASES DE DATOS

a. El equipo de desarrollo genera la estructura interna de las bases de datos.
b. El equipo responsable del diseño de la arquitectura de *software* o quien cumpla ese rol:

[4] http://www.databaseanswers.org/data_models/

- Revisa y aprueba los documentos de diseño elaborados por el equipo de desarrollo.
- Brinda orientación a los integrantes del equipo de desarrollo, en lo que concerniente al diseño del modelo físico.
- Evalúa y autoriza el uso de procedimientos almacenados.
- Acompañado del líder técnico o quien haga sus veces en el equipo de desarrollo, identifican y proponen los patrones de diseño de base de datos autorizado.

c. El equipo responsable de implementar y administrar la infraestructura tecnológica asegura que la base de datos de producción y certificación se encuentren homologadas con la base de datos de desarrollo.
d. El responsable de la administración de base de datos realiza los cambios en las bases de datos de los ambientes de certificación y producción, siempre y cuando haya recibido de forma escrita la solicitud debidamente justificada bajo el formato vigente de actualización.

7.3.4. PRUEBAS DE CONCEPTO PARA EL AFINAMIENTO DEL DISEÑO DE LA BASE DE DATOS

El equipo de desarrollo:

a. Considera como actividad obligatoria la ejecución de la prueba de concepto de base de datos.
b. Realiza una prueba de concepto del diseño a través de una simulación lo más real posible generando datos, de acuerdo a las características del negocio, para comprobar que el diseño de la base de datos es idóneo. Como resultado de esta actividad, el diseño de la base de datos se puede ir ajustando hasta obtener un diseño óptimo.
c. Realiza las siguientes actividades en las pruebas de concepto:
   - A partir del modelo físico genera una versión de base de datos.
   - Basado en los diagramas de actividades, modelos de procesos, diagramas de flujo de datos, y por cada caso de uso y/o historias de usuario, el equipo de desarrollo inserta registros en las tablas o colecciones de la base de datos generada en el paso anterior.
   - Comprueba que el diseño responda en términos de estructura al requerimiento de información del usuario, caso contrario repita el primer paso y continuar.
d. Considera que un motivo de ajuste en el diseño de la base de datos tiene lugar en las primeras etapas de la puesta en producción de la aplicación donde se debe analizar la información obtenida por las herramientas de observabilidad de la aplicación y de la base de datos (archivos *log. Transaction log*[5] o la herramienta *Data Collector* en el caso de *SQL Server*) con el fin de afinar la operación de las bases de datos.

7.4. DESARROLLO ESTANDARIZADO DE LAS INTERFACES DE PROGRAMACIÓN DE APLICACIONES REST DEL MINEDU

7.4.1. ELABORACIÓN Y DOCUMENTACIÓN DE LA *API REST*

En base a lo descrito en el **Anexo Nº 05 de la presente Directiva**:

[5] La versión comunitaria de MongoDB con contiene log de transacciones.

a. El equipo responsable del diseño de la arquitectura de *software* o quien cumpla este rol, brinda orientación a los integrantes del equipo de desarrollo sobre:
    - Los requisitos y/o características del producto o componentes a desarrollar, evaluando las restricciones y ventajas de cada tipo de *API.*
    - Los mecanismos de seguridad de una *API* bajo el estricto cumplimiento de la normativa de seguridad vigente.
b. El equipo de desarrollo debe tomar en cuenta:
    - Consideraciones para la elaboración de las *API.*
    - Consideraciones para la documentación de las *API.*
    - Mecanismos de seguridad de las *API.*

7.4.2. PARA LA IMPLEMENTACIÓN DEL PATRÓN *API GATEWAY*

a. Se aplica una estrategia que permita ir estrangulando el producto de *software*, con el fin de generar microservicios progresivamente.
b. Finalizado el estrangulamiento y habiendo construido los microservicios, se puede implementar para cada microservicio las funcionalidades de autenticación, seguridad, *logging (tracking),* gestión de variables de entorno, entre otros.
c. Para presentar las *API* de una forma homogénea a los clientes y al mismo tiempo evitar exponer cómo está construida la estructura interna de servicios, se incorpora el patrón *API Gateway*, que se encargará de factorizar las funcionalidades mencionadas en un componente intermedio entre el cliente y las *API,* permitiendo:
    - Asegurar las *API.*
    - Obtener métricas de uso de las *API.*
    - Obtener los logs en un repositorio común.
    - Transformar peticiones y respuestas.
    - Almacenar variables de entorno.

7.4.3. PARA LA INSTALACIÓN Y CONFIGURACIÓN DEL *API GATEWAY*

a. La herramienta *Kong* es el *API Gateway* para aplicaciones *on premise* del MINEDU.
b. La herramienta *Kong* puede utilizar *PostgreSQL* como soporte de persistencia de datos.
c. La instalación de la herramienta *Kong* y la herramienta de administración *Konga* en ambiente de desarrollo se puede realizar a través de contenedores.
d. La instalación de la herramienta *Kong* en entornos productivos está sujeto a las políticas del equipo responsable de implementar y administrar la infraestructura tecnológica.
e. La administración de la instalación de *Kong y Konga* estará a cargo de:
    - Un especialista en administración del *API Gateway* de los ambientes de desarrollo.
    - Uno o más especialistas del equipo responsable de implementar y administrar la infraestructura tecnológica, quien tendrá el gobierno de los *API Gateway* de los ambientes de producción.

f. El arquitecto asignado al proyecto debe especificar la configuración y la activación de *plugins* (adicionales) en el *API Gateway* en el documento de "Arquitectura de *software*" en el capítulo denominado "Configuración de *API Gateway* y adicionales".
g. El arquitecto asignado debe definir los *plugins* (adicionales del *Kong*) a considerar en la arquitectura de *software*.
h. Para mayor detalle consultar el documento "Manual de Instalación y Configuración de *Kong"* proporcionado por el arquitecto asignado al proyecto.

7.5. SEGURIDAD DE LA INFORMACIÓN EN EL DESARROLLO DE *SOFTWARE* EN EL MINEDU

7.5.1. DE LOS PRINCIPIOS DE SEGURIDAD

Desde la concepción y durante todo el ciclo de vida del software se debe tomar en consideración los siguientes principios de seguridad abstraídos de la NTP-ISO/IEC 27001:2014 que permitirán establecer niveles de seguridad apropiados para los activos de información:

a. **Proporcionalidad:** Ajustar la estrategia de seguridad a la magnitud del riesgo y teniendo en cuenta las limitaciones practicas impuestas por la misión y el medio ambiente (o contexto).
b. **Cobertura:** Identificar y contabilizar los sistemas, actores y riesgos en el medioambiente que se desenvuelve el sistema informático.
c. **Oportunidad:** Tomar ventaja de las relaciones de los actores, recursos críticos y oportunidades estratégicas disponibles en el medioambiente que se desenvuelve el sistema informático.
d. **Rigor:** Especificar y obligar a que los estados esperados, comportamientos y procesos que gobiernan a los sistemas y actores relevantes se comporten como fueron diseñados.
e. **Reducción de oportunidades de ataque:** Minimizar el tamaño, cantidad y complejidad de las funciones que deberán ser protegidas limitando la exposición al ataque.
f. **Compartimentación:** Aislar los elementos del sistema y habilitar el control de interacción solo a los estrictamente necesarios para el propósito específico.
g. **Tolerancia a fallas:** Anticipar y prepararse para el potencial compromiso y falla de los sistemas y/o controles de seguridad.

7.5.2. OBJETIVOS Y CONTROLES DE SEGURIDAD

a. Se debe establecer objetivos de seguridad desde la concepción de un producto de *software* y durante todo el ciclo de vida del mismo, lo que permitirá dar soporte a la gestión de la seguridad del sistema informático.
b. Para lograr claramente la identificación de los controles de seguridad a ser aplicados en el *software* a desarrollar se debe definir los objetivos y controles de seguridad listados en la normativa vigente y actual, así como los identificados durante las pruebas de *pentesting* de seguridad a los que serán sometidos los desarrollos de sistemas informáticos.
c. Los objetivos y controles de seguridad seleccionados deben estar en concordancia con el apetito del riesgo establecido en el inicio del proyecto.

d. Los objetivos de control y los controles de seguridad mínimos indispensables que deben cumplirse de la NTP-ISO/IEC 27001:2014 (actual vigente) son*:*
   - *A*.12.1.4 Separación de entornos de desarrollo, pruebas y operaciones.
   - A.14.1.2 Aseguramiento de servicios de aplicaciones sobre redes públicas.
   - A.14.2.1 Política de desarrollo seguro.
   - A.14.2.6 Ambiente de desarrollo seguro.
   - A.14.2.8 Pruebas de seguridad del sistema.
   - A.14.2.9 Pruebas de aceptación del sistema.

7.5.3. POLÍTICA DE DESARROLLO SEGURO

La política de desarrollo seguro se aplicará en todos sus extremos según lo indicado en el numeral 7.9.2 de la Directiva para la Gestión de la Seguridad de la información del Ministerio de Educación, aprobada por Resolución de Secretaría General N° 148-2021-MINEDU.

7.5.4. ASEGURAMIENTO DE SERVICIOS DE APLICACIONES SOBRE REDES PÚBLICAS

a. La información involucrada en un *software* desarrollado que pase o no a través de redes públicas debe ser protegida contra actividades de fraude, disputa de contratos, alteración o modificación no autorizada, secuestro o robo de la información, verificando los accesos a la información mediante mecanismos de autorización y autenticación.
b. Implementar controles de seguridad que den cumplimiento a la normativa vigente que el Estado peruano establece respecto a la protección de datos.
c. El ingreso, procesamiento y edición de datos debe ser ejecutado completamente, con exactitud y de manera oportuna.
d. La información clasificada como sensible, confidencial, secreta y ultra secreta debe ser protegida durante su recolección, procesamiento y almacenamiento.

7.5.5. PRUEBAS DE SEGURIDAD DEL SOFTWARE

a. Todo desarrollo de *software* se someterá a pruebas y verificaciones de seguridad de la información.
b. Las pruebas de seguridad o *pentesting* deben ser aplicadas durante el desarrollo, integración y durante su servicio en el ambiente de producción.
c. Las pruebas de seguridad deben ser ejecutadas durante el desarrollo (según las actividades de la metodología de desarrollo adoptada o marco de trabajo), también deben ser ejecutadas durante sus pases a certificación (o evaluaciones de seguridad). En esta etapa deben ejecutarse pruebas de seguridad integrales en todos los módulos o componentes del sistema informático evaluado.
d. Respecto a las pruebas de seguridad en ambientes de producción, éstas deben programarse con el debido tiempo y detalle, a fin de no impactar en la información productiva y los servicios que se ofrecen.
e. En caso de que el equipo responsable de las pruebas de seguridad encuentre errores y/o truncantes, deben identificar flujos alternos para

lograr encontrar la mayor cantidad de bugs de seguridad, en caso de necesitar, deben comunicarse con el equipo de desarrollo a fin de solicitar una capacitación sobre el modo de configuración o flujos, previa revisión de la documentación proporcionada, antes de devolver el sistema, para evitar sucesivas iteraciones.

f. Para el proceso de pruebas de seguridad integrales, previamente revisar la documentación proporcionada por el equipo de desarrollo, a fin de que las pruebas sean óptimas y mantener el alcance del mismo.

g. Las pruebas de seguridad del *software* deben ser realizadas en un ambiente separado, de tal forma que permita garantizar que no se introducirán vulnerabilidades al ambiente de certificación y asegurar que las mismas sean confiables, para ello se podrán utilizar herramientas automatizadas como las de análisis de código, escáneres de vulnerabilidades entre otras, tal como se ha indicado en el **Anexo Nº 02 de la presente Directiva,** así como herramientas personalizadas según requieran los escenarios que son evaluados por el equipo de seguridad.

7.5.6. SEPARACIÓN DE ENTORNOS DE DESARROLLO, PRUEBAS Y OPERACIONES

El equipo responsable de diseñar, implementar y administrar la infraestructura tecnológica asegura que los ambientes de desarrollo, certificación y producción están implementados y configurados en segmentos separados de forma física y lógica.

8. RESPONSABILIDADES

8.1. La OTIC es responsable de velar por el estricto cumplimiento y seguimiento de las disposiciones de la presente Directiva, así como absolver y brindar tratamiento a los aspectos técnicos no contemplados en la misma, con observancia a la normatividad vigente.

8.2. La OTIC es responsable de actualizar la presente Directiva y/o sus anexos conforme a las necesidades institucionales y de acuerdo a la evolución del estado del arte de las tecnologías de información.

8.3. Las unidades orgánicas dependientes de la OTIC deben asegurar la aplicación de la presente Directiva, en el marco de sus funciones y en concordancia con la estrategia institucional y necesidades de las áreas usuarias.
Todos los órganos y unidades orgánicas del MINEDU involucrados en el alcance de la presente Directiva deben cumplir con las disposiciones y consideraciones establecidas, fomentar su aplicación, la predisposición al autoaprendizaje y la capacitación continua.

9. DISPOSICIONES COMPLEMENTARIAS

Las disposiciones contenidas en la presente Directiva son de aplicación obligatoria para la construcción de nuevos productos de *software* del MINEDU.

10. ANEXOS

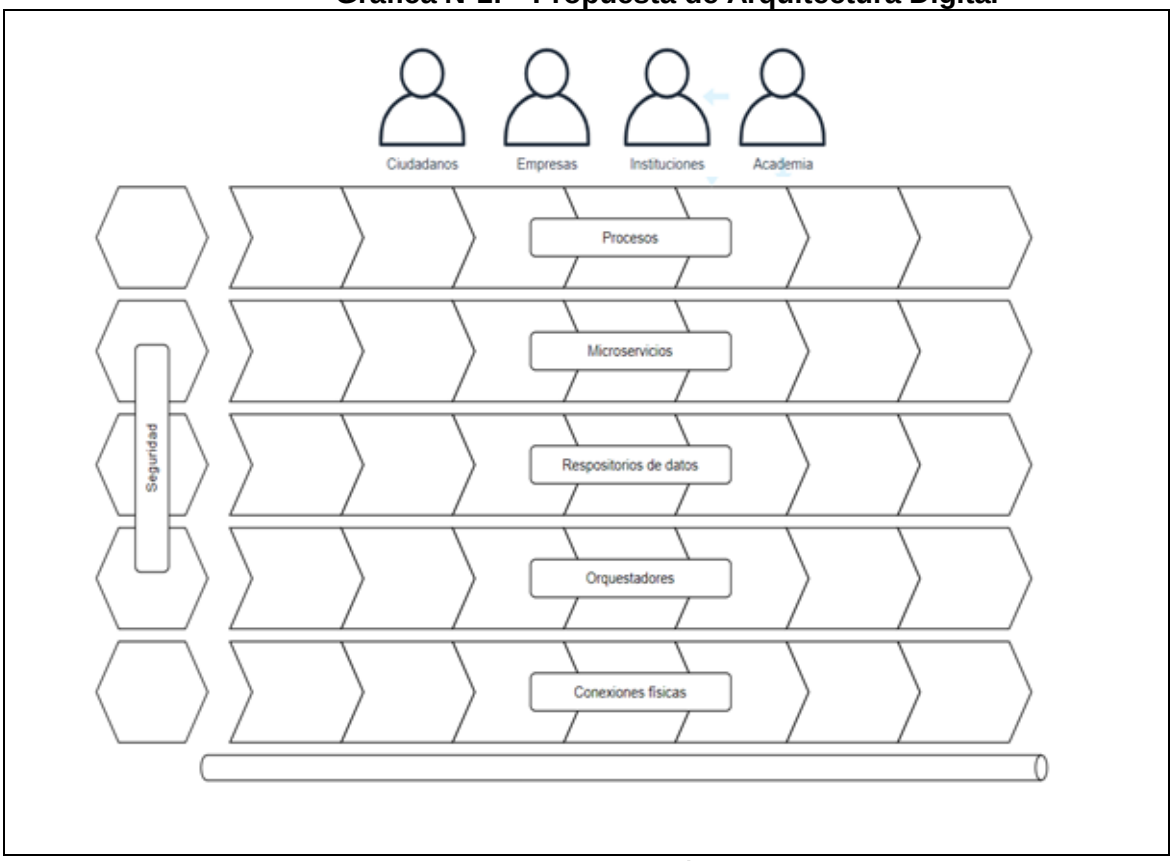
- **Anexo Nº 01**: Propuesta de Arquitectura Digital desde la perspectiva del Sector Educativo.
- **Anexo Nº 02**: Estandarización de Herramientas Tecnológicas, Principios y Patrones para el desarrollo de *software*.
- **Anexo N° 03**: Especificaciones y Consideraciones Técnicas para el uso de repositorios de código fuente.
- **Anexo Nº 04:** Especificaciones y Consideraciones Técnicas para el diseño y uso de bases de datos *SQL* y *NOSQL*.
- **Anexo Nº 05**: Especificaciones y Consideraciones Técnicas para el uso de Interfaces de Programación de Aplicaciones.

ANEXO Nº 01

PROPUESTA DE ARQUITECTURA DIGITAL DESDE LA PERSPECTIVA DEL SECTOR EDUCATIVO (v1.0)

La siguiente gráfica muestra la propuesta de visión de la arquitectura digital desde la perspectiva del sector educativo, con el fin de lograr objetivos concretos dentro de un ecosistema intersectorial.

Gráfica Nº1: Propuesta de Arquitectura Digital



Fuente: Elaboración propia

ANEXO Nº 02

ESTANDARIZACIÓN DE HERRAMIENTAS TECNOLÓGICAS, PRINCIPIOS Y PATRONES PARA EL DESARROLLO DE SOFTWARE (v1.0)

1.1 DEFINICIONES

- ***Back-end*:** Es la capa de cada sistema informático que se encarga del acceso a los datos. El *back-end* no es directamente accesible por los usuarios y además contiene la lógica de la aplicación.
- ***Framework*:** Es un conjunto de módulos que permiten el desarrollo ágil de aplicaciones mediante el uso de librerías y/o funcionalidades ya creadas.
- ***Front-end***: Es parte visible del desarrollo *web* o del lado del usuario final, que interactúa con los usuarios, es conocida como el lado del cliente. Son aquellas tecnologías de diseño y desarrollo *web* que se ejecutan en el navegador y que se encargan de la interactividad con los usuarios como por ejemplo botones, menús, páginas, enlaces, gráficos y otros componentes de una página.
- **Guías de estilo**: Son un conjunto de convenciones de cómo escribir código fuente en un lenguaje de programación en particular.
- ***HTTP*:** *Hypertext Transfer Protocol* (en español Protocolo de Transferencia de Hipertexto). Es un protocolo de comunicación de *Internet* que permite las transferencias de información a través de archivos en la *World Wide Web* y que funciona por peticiones y respuestas. Es un protocolo que no está basado en el estado, lo cual significa que el protocolo trata cada petición como una transacción independiente y no tiene en cuenta cualquier solicitud anterior.
- ***HTTPS*:** *Hypertext Transfer Protocol Secur*e (en español Protocolo Seguro de Transferencia de Hipertexto). Es un protocolo de comunicación de Internet que protege la integridad y la confidencialidad de los datos, entre los equipos de cómputo de los usuarios y el sitio *web.*
- **IDE:** *Integrated Development Environment (*en español Entorno de Desarrollo Integrado). Es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de *software*.
- ***OWASP Testing Guide:*** Manual de operaciones de seguridad, que aglutina las más importantes buenas prácticas de seguridad en las aplicaciones *web*.
- **Patrón:** En el marco del desarrollo de sistemas, patrón es una solución general reusable que puede ser aplicada a problemas que ocurren comúnmente en el desarrollo de *software*. Es la descripción o plantilla de cómo resolver un problema que puede ser usado en diferentes situaciones.
- **Principio:** Es una afirmación que no requiere justificación por sí misma, que suele hacerse evidente por la observación de fenómenos comparables y que son independientes del momento histórico, la cultura o la geografía.
- **Recurso:** En el marco de los sistemas de información distribuidos, es cualquier elemento de información distinguible y disponible en internet. Por ejemplo, un documento electrónico, una imagen, un servicio, etc.
- **Repositorio de objetos:** Es utilizado como almacenamiento de elementos binarios de cualquier tipo, formato o tamaño. Es una arquitectura de almacenamiento de datos, manejándolos como objetos, al contrario de otras arquitecturas de almacenamiento como los sistemas de archivo, que manejan datos como una jerarquía de archivos.
- ***UML:*** *Unified Modeling Language* (en español Lenguaje Unificado de Modelado) fue creado para forjar un lenguaje de modelado visual común y semántica y

sintácticamente rico para la arquitectura, el diseño y la implementación de sistemas de software complejos, tanto en estructura como en comportamiento.

1.2 *DEVSECOPS*

La seguridad de la información es una característica muy importante en el diseño de un producto de *software.* Es por ello que MINEDU adopta *DEVSECOPS* para integrar la seguridad en los procesos de desarrollo y operaciones de los sistemas de información. Este modelo consiste en llevar a cabo el desarrollo de un servicio o *software* considerando la seguridad desde el inicio del proceso. A continuación, se describen algunos de los beneficios inherentes a este enfoque:

- Temprana identificación de vulnerabilidades en el código.
- Mayor velocidad y agilidad en la aplicación de la seguridad.
- Habilidad para responder a los cambios y requisitos rápidamente.
- Mejor colaboración y comunicación entre equipos.
- Más oportunidades para desarrollos automatizados y pruebas de calidad.
- Concientización sobre la seguridad de la información entre los equipos.

1.2.1 ACTIVIDADES ASOCIADAS A DEVSECOPS

Las siguientes actividades deben ser ejecutadas de forma permanente durante el ciclo de vida del *software*:

a. **Revisión y Análisis de código**: Consiste en entregar el código de forma incremental, con el objetivo de poder detectar las vulnerabilidades rápidamente.
b. **Gestión de cambios**: Consiste en incrementar la velocidad y eficiencia permitiendo que los cambios vengan de cualquier fuente, para luego determinar si estos cambios son beneficiosos o no mediante un proceso de revisión.
c. **Monitorización de cumplimiento**: Consiste en que el equipo debe estar listos para una auditoría en cualquier momento.
d. **Investigación de amenazas**: Consiste en identificar posibles amenazas con cada actualización del código y ser capaz de responder a tiempo de manera eficiente.
e. **Análisis de vulnerabilidades**: Consiste en identificar nuevas vulnerabilidades con análisis de código, *pentesting* o *hacking ético* y análisis de arquitectura, para luego analizar los tiempos de respuesta.
f. **Formación en seguridad**: Consiste en formar en buenas prácticas de seguridad a los equipos de desarrollo.

1.3 PRINCIPIOS DE DESARROLLO DE *SOFTWARE*

Recursos de Consulta:

○ Principios *SOLID*	https://www.baeldung.com/solid-principles
○ Principios de Diseño Orientado a Objetos)	http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod
○ Diseño guiado por el dominio (DDD)	https://bit.ly/39GSzZW
○ Desarrollo guiado por pruebas *(*TDD)	https://es.wikipedia.org/wiki/Desarrollo_guiado_por_pruebas

1.4 PRINCIPIOS DE SEGURIDAD EN *DEVSECOPS*

Recursos de Consulta

○ DevSecOps	https://www.redhat.com/es/topics/devops/what-is-devsecops https://www.eccouncil.org/ https://owasp.org/www-project-devsecops-guideline/ https://owasp.org/www-project-devsecops-maturity-model/ Tomar en consideración los controles de la ISO/IEC 27017 y las indicaciones de seguridad de Cloud Security Alliance.

1.5 PRINCIPIOS PARA LA SEGURIDAD DE LA INFORMACIÓN

Recursos de Consulta:

○ ISO/IEC 27001 Sistemas de Gestión de la Seguridad de la Información	https://www.isotools.org/normas/riesgos-y-seguridad/iso-27001/ https://www.isotools.org/normas/riesgos-y-seguridad/iso-27001/
○ OWASP Testing Guide	https://dsomm.timo-pagel.de/index.php
○ OWASP Mobile Security Testing Guide	https://owasp.org/www-project-mobile-security-testing-guide/

1.6 PATRONES DE ARQUITECTURA

Recursos de Consulta

○ Cinco patrones de arquitectura	https://bit.ly/39JetMo
○ Towards data science	https://bit.ly/36Akjxv
○ Artículo en medium	https://bit.ly/3mDnOJ6

1.7 PATRONES DE DISEÑO DE *SOFTWARE*

Recursos de Consulta

○ Patrones de diseño genéricos	*Refactoring* https://refactoring.guru/design-patterns
○ Patrones de diseño en *javascript*	*DotFactory*: https://www.dofactory.com/javascript/design-patterns

1.8 PATRONES DE DESARROLLO DE MICROSERVICIOS[6]

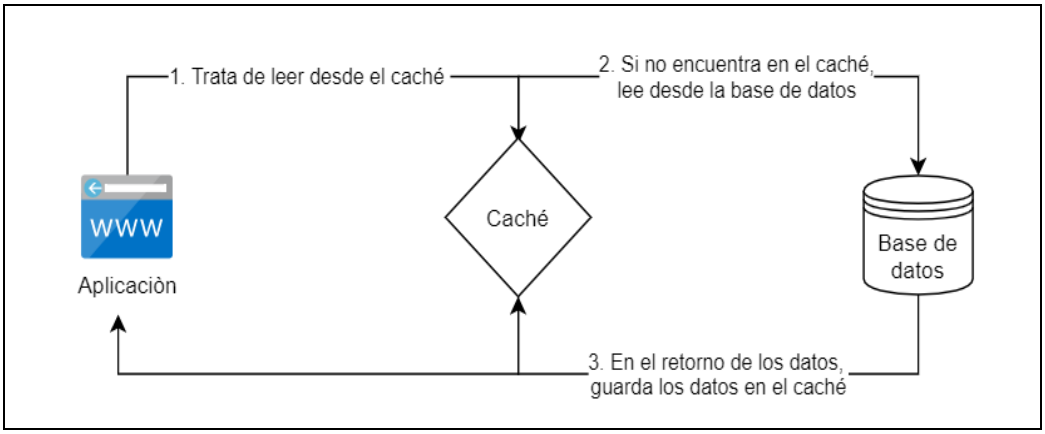
Recursos de Consulta

○ El lenguaje de patrones	https://microservices.io/
○ DotFactory:	https://bit.ly/3qpXncc https://www.dofactory.com/javascript/design-patterns

1.9 PATRONES EN EL DISEÑO DE BASE DE DATOS

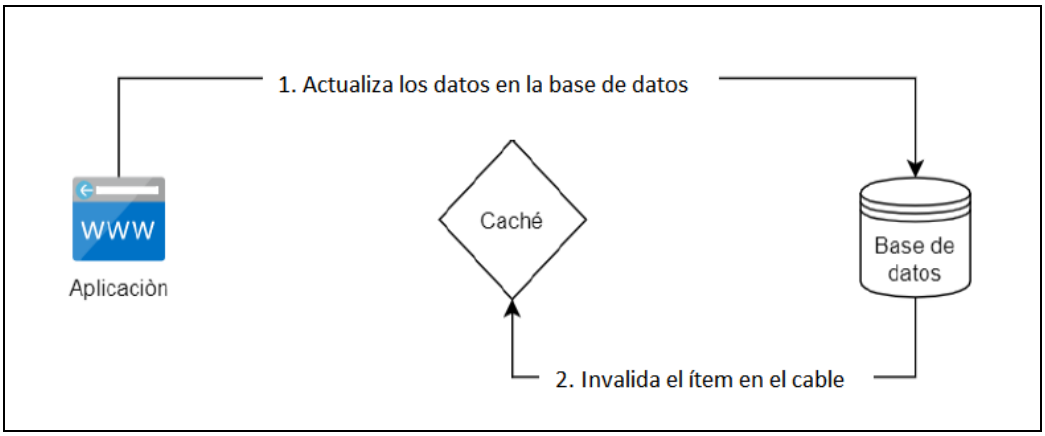
○ Patrón *Caché-Aside*	Este patrón permite mejorar los tiempos de respuesta, pues si se necesita datos específicos, primero se trata de conseguirlos en la caché. Si los datos no están allí, entonces recién se accederá a la base de datos, y en el retorno, se almacenarán los datos en el caché. Así mismo, después de cada actualización de la base datos, los datos asociados en el caché son eliminados.

Gráfica Nº1: Patrón *Cache Aside* - Creación



Fuente: Elaboración propia

Gráfica Nº2: Patrón *Cache Aside* - Actualización

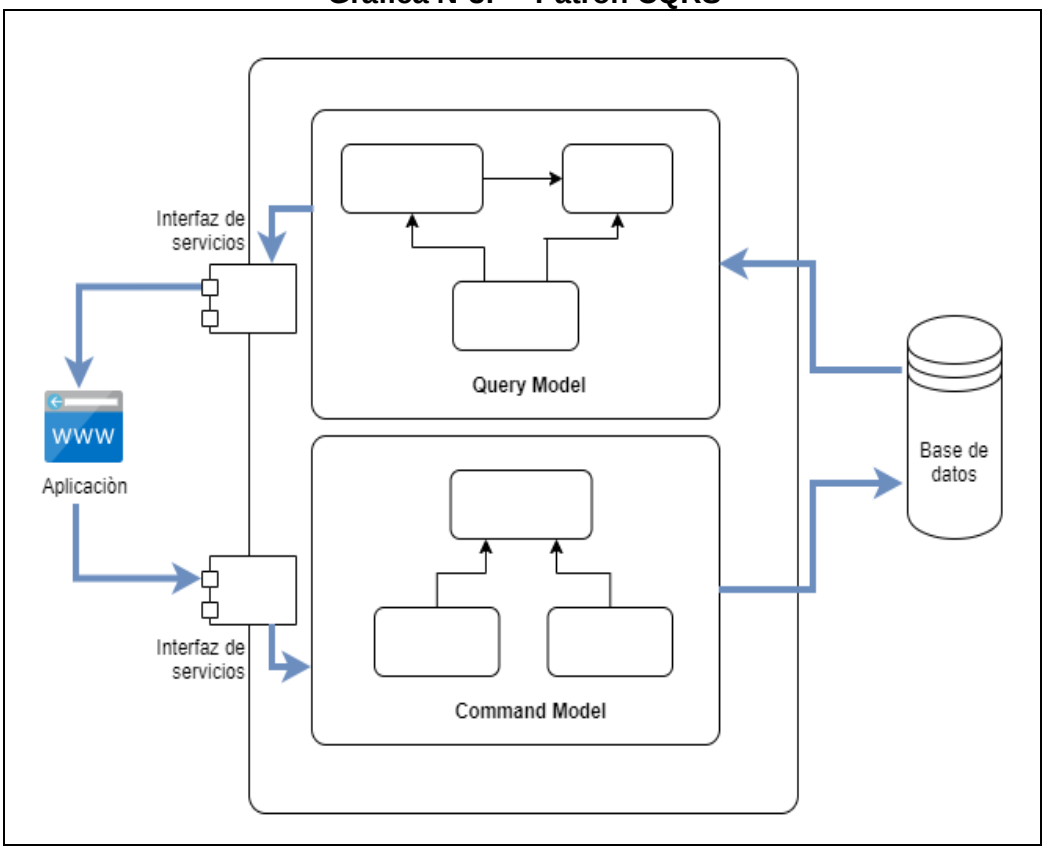


Fuente: Elaboración propia.

[6] Se considera a Chris Richardson un referente mundial en el diseño de microservicios.

○ Patrón CQRS – (*Command Query Responsability Segregation*)	Es un patrón de arquitectura que implementa dos subsistemas diferenciados, uno responsable de los comandos, y otro responsable de las consultas, donde un comando es una petición por parte del usuario u otro sistema para realizar una operación de actualización que cambie el estado de un sistema. El subsistema de consulta recibe los cambios en el estado del sistema mediante un mecanismo de sincronización.

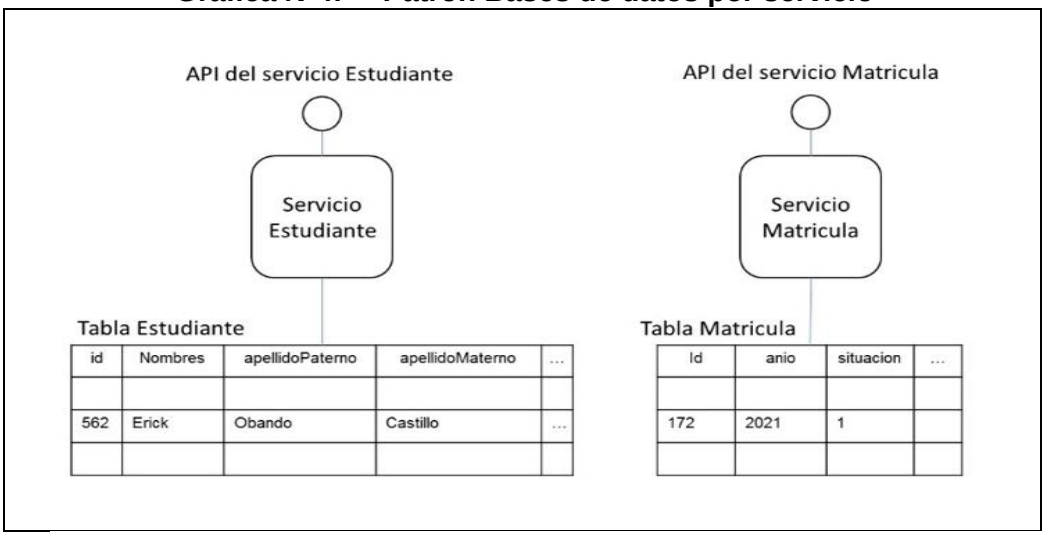
Gráfica Nº3: Patrón CQRS



Fuente: Elaboración propia.

○ Patrón Bases de datos por servicio	Cuando se desarrolla una aplicación con el patrón de arquitectura de microservicio Los servicios deben persistir la data en algún tipo de base de datos.

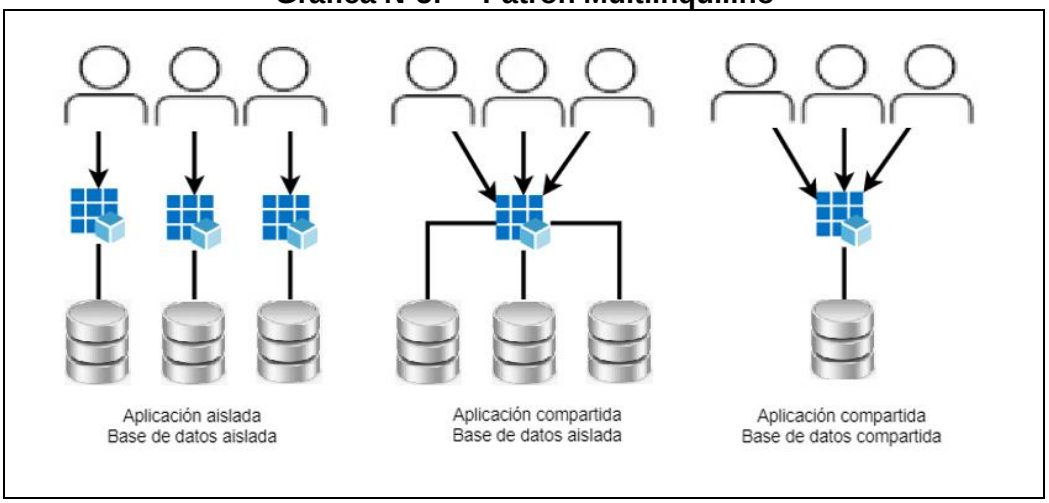
Gráfica Nº4: Patrón Bases de datos por servicio



Fuente: Elaboración propia.

○ Patrón Multiinquilino (*Multitenancy*)	Este es un patrón en el cual una instancia de una aplicación es usada por diferentes clientes, acortando los tiempos de desarrollo. Hay múltiples estrategias para implementar este patrón, que va desde las implementaciones de inquilinos altamente aisladas hasta las altamente distribuidas.

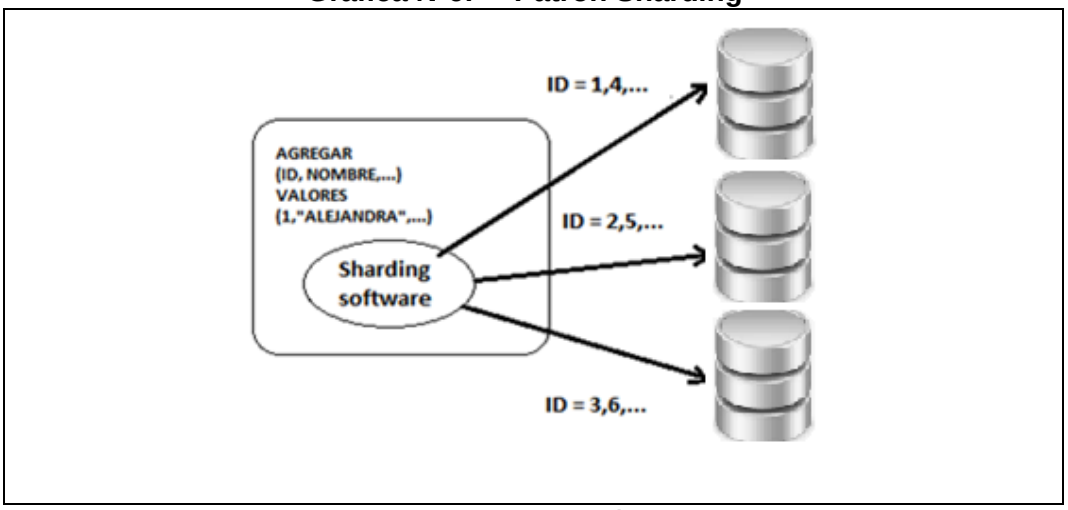
Gráfica Nº5: Patrón Multiinquilino



Fuente: Elaboración propia

○ Patrón Particionamiento (*Sharding*):	Este patrón permite dividir las bases de datos en un conjunto horizontal de particiones o *shards* con el fin de mejorar la escalabilidad cuando se trata de almacenar y acceder a grandes volúmenes de datos.

Gráfica Nº6: Patrón Sharding



Fuente: Elaboración propia.

1.10 MARCOS Y MODELOS PARA EL DESARROLLO DE PRODUCTOS DE *SOFTWARE*

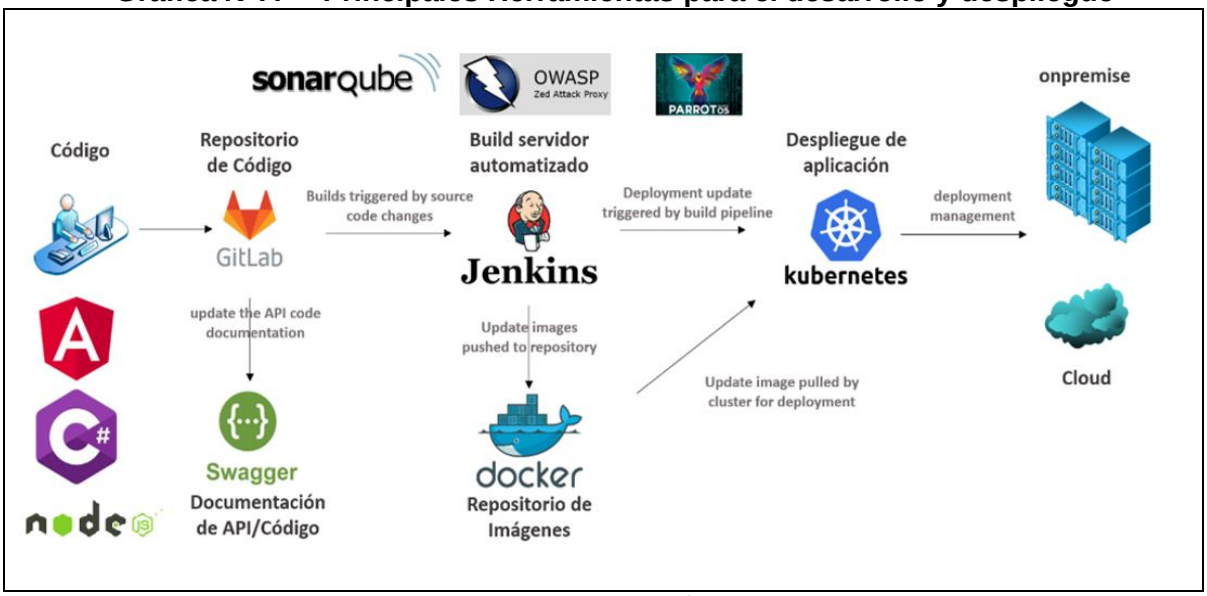
Modelo en Cascada	Modelo que ordena rigurosamente las etapas del ciclo de vida del *software*, de tal forma que el inicio de cada etapa debe esperar a la finalización de la inmediatamente anterior. Este modelo considera las siguientes etapas: análisis, diseño, implementación, prueba y despliegue. • Este modelo se desempeña bien en proyectos con requisitos claros o cuando se trabaja con herramientas técnicas. • Se utiliza para proyectos que son rígidos, y además donde se especifiquen muy bien los requerimientos y se conozca muy bien la herramienta a utilizar. • Facilita la división de tareas en roles (personas) **Referencia:** https://es.wikipedia.org/wiki/Desarrollo_en_cascada
Modelo Iterativo Incremental	Modelo que se basa en varios ciclos cascada realimentados aplicados repetidamente, con una filosofía iterativa. • Reduce el tiempo de desarrollo inicial, ya que se implementa la funcionalidad parcial. • Proporciona todas las ventajas del modelo en cascada realimentado, reduciendo sus desventajas sólo al ámbito de cada incremento. • Se basan en la retroalimentación sobre los avances. • Resulta más sencillo acomodar cambios al acotar el tamaño de los incrementos. **Referencia:** https://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente

Extreme Programming (XP)	Marco de trabajo para el desarrollo ágil que tiene por objetivo producir *softwar*e de mejor calidad. *XP* es apropiado para los siguientes escenarios: • Requisitos de *software* que cambian dinámicamente. • Riesgos provocados por desarrollos de tiempo fijo que utilizan nueva tecnología. • Equipo de desarrollo extendido pequeño y co-ubicado. • La tecnología que está utilizando permite realizar pruebas funcionales y unitarias automatizadas. **Referencia**: https://www.agilealliance.org/glossary/xp
Scrum	Es un marco de trabajo adaptable, iterativo, rápido, flexible y eficaz, diseñado para ofrecer un valor considerable en forma rápida a lo largo del proyecto. que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptables para problemas complejos. Está conformado por un conjunto de valores, principios y prácticas ligeras que se basan en la conformación de equipos multifuncionales para entregar productos y servicios en ciclos cortos, lo que permite: • Retroalimentación rápida • Mejora continua • Adaptación rápida al cambio y • Entrega acelerada **Referencia:** https://scrumguides.org

1.11 HERRAMIENTAS DE DESARROLLO DE *SOFTWARE*

Para la implementación de *DEVSECOPS* en el MINEDU se debe seleccionar herramientas tecnológicas que permitan obtener mayor eficiencia, productividad y agilidad en el proceso integrado de desarrollo, pruebas y puesta en producción de los productos de *software*. A continuación, una gráfica con las principales herramientas para el desarrollo y despliegue:

Gráfica Nº7: Principales Herramientas para el desarrollo y despliegue



Fuente: Elaboración propia

1.11.1 HERRAMIENTA DE DIAGRAMACIÓN Y MODELAMIENTO

A. **Modelamiento de procesos:**

***Bizagi*:** Es una *suite* ofimática con dos productos complementarios, un modelador de procesos y una *suite* de *BPM.* Esta herramienta de mapeo de procesos de negocio gratuito, intuitivo y colaborativo. *Bizagi* permite a las organizaciones crear y documentar los procesos de negocio en un repositorio central en la nube para obtener un mejor entendimiento de cada paso e identificar las oportunidades de mejora de los procesos para aumentar la eficiencia de la organización.

Referencia: https://www.bizagi.com/es

B. Modelamiento de objetos:

a. ***Power Designer*:** Es una herramienta de modelado empresarial colaborativa que permite visualizar, analizar y manipular de manera más fácil los metadatos para tener una arquitectura de información de empresa eficaz. Esta herramienta cuenta con diferentes técnicas de modelización (modelo conceptual tradicional, físico y lógico con una modelización única de inteligencia de negocios y de traslado de datos).

Referencia: https://www.powerdesigner.biz/ES/

b. ***Drawio*:** Es una herramienta gratuita de diagramación que permite la creación de diagramas, contando con modelos para diversos tipos como pueden ser diagramas *UML*, esquemas de red, flujogramas, diagramas de ingeniería y electrónica, mapas conceptuales, diagramas de *Venn*. También permite crear colecciones de diagramas e imágenes personalizados para utilizar en los diagramas.

Referencia: https://www.draw.io

C. Modelamiento de base de datos

Power Designer: Es una herramienta de modelado empresarial colaborativa que permite visualizar, analizar y manipular de manera más fácil los metadatos para tener una arquitectura de información de empresa eficaz. Posee técnicas de modelización como modelo conceptual tradicional, físico y lógico con una modelización única de inteligencia de negocios y de traslado de datos.

Referencia: https://www.powerdesigner.biz/ES/

D. Prototipo

a. ***Figma*:** Es un editor de gráficos vectorial y una herramienta de generación de prototipos, principalmente basada en la *web,* con características *off-line* adicionales habilitadas por aplicaciones de escritorio en sistemas operativos *macOS* y *Windows*.

Referencia: https://www.figma.com

b. ***Camtasia***: Es una herramienta que permite grabar el audio y vídeo de nuestro escritorio. Permite personalizar los vídeos con música (tanto propia como de su biblioteca libre de derechos) y añadir transiciones cuando haya cortes en los vídeos.

Referencia: https://www.techsmith.com/video-editor.html

c. ***Balsamiq*:** Es una herramienta que permite crear prototipos, bocetos o *wireframes*. Permite escoger entre diferentes objetos prediseñados como: barras de estado, menús, barras de progreso, entre otros. Además, permite exportar el diseño en formatos *PNG, PDF* e incluso al portapapeles.

Referencia: https://balsamiq.com

d. ***Pencil:*** Es una herramienta multiplataforma con el que se pueden elaborar maquetas y diseñar la interfaz gráfica de aplicaciones.

Referencia: https://pencil.evolus.vn/

1.11.2 FRAMEWORKS DE DESARROLLO

a. ***Flutter*:** De código abierto desarrollado por *Google* para crear aplicaciones móviles nativas. Genera código 100% nativo para cada plataforma, por lo que el rendimiento y la experiencia de usuario es totalmente idéntico a las aplicaciones nativas tradicionales.

Recursos de Consulta:

○ Tutorial for Beginners (35 sessions)	https://bit.ly/3lDZyW3
○ Página official	https://flutter.dev/
○ Flutter (84 sessions)	https://bit.ly/3qoAwhb

b. ***Javascript*:** Permite construir componentes tanto en el *back-end* como en el *front-end*. Asimismo, es posible desarrollar aplicaciones hibridas, ya sea aplicaciones para equipos móviles o aplicaciones *web*. Las siguientes son las herramientas a considerar:

○ Angular (Google):	https://angular.io/docs Tutorial (7 videos): https://bit.ly/2JOKRSD Angular avanzado: https://angular-university.io/
○ ReactJS (Facebook):	https://es.reactjs.org https://es.reactjs.org/tutorial/tutorial.html https://egghead.io/courses/the-beginner-s-guide-to-react
○ Tutorial	https://codingpotions.com/angular-material

c. ***.Net:*** Es un *framework* informático administrado, gratuito y de código abierto para los sistemas operativos *Windows, Linux y MacOS*. Es sucesor multiplataforma de *.NET Core.*

Recursos de Consulta:
https://docs.microsoft.com/en-us/dotnet/
https://dotnet.microsoft.com/learn/aspnet/microservices-architecture

d. ***NodeJS*:** Es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa de servidor basado en el lenguaje de programación *JavaScript,* asíncrono, con E/S de datos en una arquitectura orientada a eventos y basado en el motor V8[7] de Google.

Los *frameworks* de desarrollo basados en *NodeJS* a considerar son:

○ NestJS Framework	https://docs.nestjs.com/
○ ExpressJS	https://expressjs.com/es/

e. ***Spring Boot*:** Es uno de los *frameworks* más utilizados por la comunidad y cuenta con herramientas que facilitan el desarrollo de aplicaciones.

Recursos de Consulta:

○ Spring Boot Microservices Level 1: Communication and Discovery	https://bit.ly/3oltPur
○ Spring Boot Microservices Level 2: Fault tolerance and resilience (Eureka, Hystrix)	https://bit.ly/2JzPpwr

[7] V8 es un motor *open source* escrito en C++ para compilar *JavaScript* y *WebAssembly* en código máquina.

○ Spring Boot Microservices Level 3: Using property file config with Spring Boot	https://bit.ly/3mIIlYv
○ Spring Data MongoDB	https://spring.io/projects/spring-data-mongodb
○ Spring Boot Security	https://bit.ly/37CbYbQ
○ Documentación	https://spring.io/projects/spring-boot
○ Tutorial	https://www.javatpoint.com/spring-boot-tutorial

1.11.3 IDE DE DESARROLLO

a. ***IntelliJ*:** Entorno de desarrollo integrado para el desarrollo de código fuente en Java. Es desarrollado por *JetBrains.* Tiene edición comunitaria y empresarial. comercial.

Referencia: https://www.jetbrains.com/es-es/idea/

b. ***SpringToolSuite4*:** Herramienta de desarrollo de codificar aplicaciones empresariales con *Spring. Integrado a Eclipse, Visual Studio Code o IntelliJ*.

Referencia: https://spring.io/tools

c. ***Visual Studio**:* Entorno de edición de código para aplicaciones multiplataforma y nube. Permite recarga activa de aplicaciones *.Net* así como la compilación, depuración y *pruebas* de aplicaciones en *Windows, Mac y Linux*.

Referencia: https://visualstudio.microsoft.com

d. ***Visual Studio Code***: (*Linux, MacOS y Windows*): Editor de código fuente de *Microsoft* que incluye soporte para depuración, integración con *GIT*, resaltado de sintaxis, fragmentos y finalización inteligente de código.

Referencia: https://code.visualstudio.com/

e. ***Sublime*:** Es un editor de texto y editor de código fuente. Está escrito en *C++* y *Python* para los *plugins*. Desarrollado originalmente como una extensión de *Vim*.

Referencia: https://www.sublimetext.com/

f. ***Notepad++:*** Es un editor de texto y de código fuente libre con soporte para varios lenguajes de programación. Con soporte nativo para *Windows.* Tiene similitud con el *bloc* de notas en cuanto al hecho de que puede editar texto sin formato y de forma simple.

Referencia: https://notepad-plus-plus.org/

g. ***SQL Server Management Studio (SSMS)*:** Es un entorno integrado para administrar cualquier infraestructura *SQL*, desde *SQL Server* hasta *Azure SQL* Database. *SSMS* proporciona herramientas para configurar, monitorear y administrar instancias de *SQL Server* y bases de datos. Se utiliza *SSMS* para implementar, supervisar y actualizar los componentes de la capa de datos que utilizan sus aplicaciones y crear consultas y scripts. Así como para consultar, diseñar y administrar bases de datos sea que estén en la nube o localmente.

Referencia: https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15

1.11.4 LENGUAJES DE PROGRAMACIÓN Y GUÍA DE ESTILO

Los siguientes son los lenguajes de programación a considerar:

○ C#	Guía de estilo (Google) https://google.github.io/styleguide/csharp-style.html
○ JavaScript	Guía de estilo (Google) https://google.github.io/styleguide/javaguide.html Guía de estilo (Google) https://google.github.io/styleguide/jsguide.html Guía de estilo (AirBnb) https://github.com/airbnb/javascript
○ TypeScript	https://www.typescriptlang.org/
○ Python	Guía de estilo (Python) https://google.github.io/styleguide/pyguide.html
○ Java	http://cr.openjdk.java.net/~alundblad/styleguide/index-v6.html
○ PHP	https://www.php-fig.org/psr/psr-2/

Referencia: https://google.github.io/styleguide/

1.11.5 GESTORES DE BASES DE DATOS

a. ***Elasticsearch*:** Es un servidor de búsqueda basado en *Lucene*. Provee un motor de búsqueda de texto completo, distribuido y con capacidad de multitenencia con una *interfaz web RESTful* y con documentos *JSON*. Está desarrollado en *java* y está publicado bajo las condiciones de la licencia *Apache*.

Referencia: https://www.elastic.co/es

b. ***Microsoft SQL Server* (>= V2017)**

Referencia: https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15

c. ***MySQL***: Es la versión de descarga gratuita de la base de datos de código abierto más popular del mundo. Está disponible bajo la licencia *GPL* y cuenta con el

respaldo de una enorme y activa comunidad de desarrolladores de código abierto Es utilizado en MINEDU en aplicaciones ya existentes y otras de prefabricadas, que requieren mantenimiento evolutivo y en aplicaciones o plataformas *open source*.

Referencia: https://www.mysql.com/products/community/

d. ***MongoDB*:** En lugar de guardar los datos en tablas, tal y como se hace en las bases de datos relacionales, *MongoDB* guarda estructuras de datos *BSON* (una especificación similar *a JSON*) con un esquema dinámico, haciendo que la integración de los datos aplicaciones sea más fácil, rápida y escalable.

Recursos de Consulta:

○ Documentación	https://docs.mongodb.com/guides/
○ MongoDb University	https://university.mongodb.com/

e. ***PostgreSQL***: Sistema de gestión de base de datos relacional orientado a objetos y de código abierto, publicado bajo la licencia *PostgreSQ*L. En el MINEDU, se considera como una alternativa en los Sistemas de Información Geográficos.

Referencias:
https://www.postgresql.org/
https://www.citusdata.com/

1.11.6 HERRAMIENTAS DE INTEGRACIÓN Y DESPLIEGUE CONTINUO

a. ***Codemagic*:** Herramienta online de integración y despliegue continuo para aplicaciones móviles creadas con *Flutter, React Native, Android, Native IOS*. Funciona enteramente online sin tener que añadir ningún tipo de configuración especial al proyecto. Con *Codemagic*, se puede ejecutar pruebas escritas para el proyecto y, si se ejecutan correctamente, compilar el código a las plataformas deseadas y obtener los artefactos derivados.

Referencia: https://codemagic.io/start/

b. ***GitLab* CI/CD:** Es una herramienta cuyo uso debe estar asociado a proyectos de construcción de sistemas de información simples y con pocas herramientas tecnológicas de despliegue: servicios, microservicios y solo un único motor de datos.

c. ***Git CLI:*** Es una herramienta de línea de comandos para ejecutar todos los comandos de *GIT.*

d. **Interfaces Gráficas GIT:** Es una herramienta que permite el uso de *GIT* de una manera más sencilla, evitando utilizar comandos en consola. Algunos de los utilizados en MINEDU, en orden de recomendación, son: *GIT Extensions, Sourcetree, Git Kraken, Visual Studio Code* y *Visual Studio.*

Recursos de Consulta:

○ Configuración de *Gitlab CI Runner*	Cómo desplegar *Registry* https://docs.docker.com/registry/deploying/ https://bit.ly/36ENpfc
○ Git Container Registry	Publicar imagen *Docker* a GCR: https://bit.ly/3lDyUfS

e. ***Jenkins*:** Servidor de automatización *open source* basado en java. Apoya en la automatización mediante integración continua y facilita el despliegue continuo. Se integra con *GIT* y su uso debe estar asociado a proyectos de construcción de sistemas de información complejos y con diversas herramientas tecnológicas de despliegue: servicios, microservicios, múltiples bases de datos, colas, contenedores, orquestadores, entre otros.

Referencia: https://www.jenkins.io/doc/

1.11.7 SERVIDORES *WEB*

a. ***Apache Tomcat / Glassfish*:** Es un *servlet Java* de código abierto y un contenedor de páginas de servidor *Java* que permite a los desarrolladores implementar una variedad de aplicaciones empresariales *Java*. *Tomcat* también ejecuta un entorno de servidor *web HTTP* en el que se puede ejecutar código *Java*. Funciona como un contenedor de *servlets* desarrollado en la *Apache Software Foundation.*

Referencia: http://tomcat.apache.org/

b. ***Apache HTTP***: El proyecto del *servidor HTTP Apache* es un esfuerzo por desarrollar y mantener un servidor HTTP de código abierto para sistemas operativos modernos, incluidos *UNIX y Windows*. Su objetivo es proporcionar un servidor seguro, eficiente y extensible que proporcione servicios *HTTP* en sincronía con los estándares *HTTP* actuales.

Referencia: https://httpd.apache.org/

c. ***Internet Information Server*:** Es un servidor *web* flexible y de uso general de *Microsoft* que se ejecuta en sistemas operartivos *Windows* para servir las páginas o archivos *HTML* que se solicitan. En el MINEDU está siendo utilizado para aplicaciones on-premise en servidores con sistema operativo *Windows*.

Referencia:
https://www.iis.net/
https://docs.microsoft.com/en-us/iis/get-started/whats-new-in-iis-10-version-1709/new-features-introduced-in-iis-10-1709

d. ***Nginx*:** Utilizado como servidor de archivos y como balanceador de carga.

Referencia: https://www.nginx.com/

e. ***Xamp*:** Es paquete de software libre, que consiste principalmente en el sistema de gestión de bases de datos *MySQL*, el servidor *web Apache* y los intérpretes para lenguajes de *script PHP y Perl*. El nombre es en realidad un acrónimo: *X, Apache, MariaDB/MySQL, PHP*. Es utilizado en ambientes de desarrollo.

Referencia: https://www.apachefriends.org/es/index.html

f. ***WampServer***: Conjunto de soluciones para el sistema operativo *Microsoft Windows*, que consiste en el servidor *web Apache, OpenSSL* para soporte *SSL*, base de datos *MySQL* y lenguaje de programación *PHP*.

Referencia: https://www.wampserver.com/en/

1.11.8 API GATEWAY

a. ***Kong:*** Herramienta API Gateway de tipo open source creada sobre NGINX que aporta con:
- Homogeneizar las diferentes *API* (tanto internas como externas).
- Factorizar la securización de las *API* para que no tengan que encargarse los microservicios.
- Obtener métricas de uso de las diferentes partes de las *API*.
- Obtener logs en un repositorio común.
- Almacenar variables de entorno.

Referencia: https://konghq.com/kong/

Recursos de Consulta:

○ API gateway pattern	https://microservices.io/patterns/apigateway.html
○ API Gateway by Chris Richardson (Video)	https://bit.ly/3lzFILr
○ Getting Started Guide Kong (Link)	https://bit.ly/36EQrjA

1.11.9 REPOSITORIO DE OBJETOS

a. ***MinIO*:** Servidor de objetos compatible con *Amazon S3* (distribuido bajo licencia de *Apache*). Puede almacenar datos no estructurados (fotos, videos, *backups*, entre otros). El tamaño máximo es 5TB.

Referencia: https://min.io/

Recursos de Consulta:

○ MinIO Object Storage (7 sessions)	https://bit.ly/33HZCxR

b. ***OpenIO*:** Es recomendable para *Big Data* e Inteligencia artificial, altamente escalable y tiene una arquitectura distribuida compatible con *S3*.

Referencia: https://www.openio.io/

1.11.10 CONTENT MANAGEMENT SYSTEM

a. ***Moodle*:** Es una herramienta de gestión de aprendizaje, o más concretamente de *Learning Content Management*, de distribución libre, escrita en *PHP*.

Referencia: https://moodle.org/?lang=es

b. ***Strapi*:** Es un *CMS* de tipo *headless*, orientado solamente al desarrollo del lado del servidor. Utiliza tecnología *NodeJS*.

Referencia: https://strapi.io/

c. ***Wordpress*:** Es una plataforma *open source* ampliamente difundida, desarrollada con *PHP* y utilizada para la creación de páginas o portales de diferentes dependencias de MINEDU.

Referencia: https://wordpress.com/es/

1.11.11 HERRAMIENTA PARA LA DOCUMENTACIÓN DE API

a. ***OpenAPI (Swagger):*** Conjunto de herramientas de software de código abierto para diseñar, construir, documentar, y utilizar servicios *web RESTful*. Incluye documentación automatizada, generación de código, y generación de casos de prueba. Las *API* se documentarán con esta herramienta. Para mayor información remitirse al numeral 1.5 "Consideraciones para la documentación de la *API Rest*" **del Anexo Nº 05 de la presente Directiva**.

Referencia: https://swagger.io/

1.11.12 BUS DE MENSAJERÍA Y EVENTOS

a. ***Apache Kafka*:** Es una aplicación multiplataforma especializada en el procesamiento de flujos de datos. Originariamente desarrollado por *LinkedIn*, en una plataforma de transmisión de alto rendimiento y gran potencia utilizada por *Netflix, Microsoft o Airbnb*.

Referencia: https://kafka.apache.org/

b. ***RabbitMQ*:** *Broker* de mensajería o gestor de colas. *Software* donde se pueden definir colas, las aplicaciones se pueden conectar a dichas colas y transferir / leer mensajes en ellas.

Referencia: https://www.rabbitmq.com/

1.11.13 AUTENTICACIÓN Y AUTORIZACIÓN

a. ***e Passport***: El MINEDU cuenta con una solución de desarrollo *in-house* denominada Passport versión 4 la cual permite gestionar la autenticación y autorización de usuarios mediante la aplicación de usuario y contraseña, también gestiona la autorización por roles y acciones. El sistema *Passport* pone a su disposición diferentes servicios *web* con sus funcionalidades principales. En el contexto de la atención de necesidades que requieran la aplicación de procedimientos *SSO* se debe utilizar *software* que incorporen estándares abiertos tales como *OIDC y OAuth2*, entre otras características.

b. ***Keycloak*:** Es un *software* de código abierto que permite el inicio de sesión único (*IdP*). Es compatible de forma predeterminada con los protocolos de federación

de identidad *SAML v2* y *OpenID Connect (OIDC) / OAuth2*. La intención de la herramienta es facilitar la protección de aplicaciones y servicios con poca o ninguna codificación. Un *IdP* permite que una aplicación delegue su autenticación.

Referencia: https://www.keycloak.org/

1.11.14 *BIG DATA*

a. ***Apache Beam*:** Es un modelo de programación unificada de código abierto para definir y ejecutar canales de procesamiento de datos, incluidos *ETL*, procesamiento por lotes y flujo.

Referencia: https://beam.apache.org

b. ***Apache Spark*:** Motor de procesamiento distribuido responsable de orquestar, distribuir y monitorizar aplicaciones que ejecutan de múltiples tareas de procesamiento de datos sobre varias máquinas de trabajo, que forman un clúster.

Referencia: https://spark.apache.org/

Recursos de Consulta:

○ Mastering Apache Spark (26 sessions)	https://bit.ly/36BfZhs
○ Apache Spark Tutorials (21 sessions)	https://bit.ly/3lHyVzo

c. ***Data lake*:** Es un repositorio de almacenamiento que contienen una gran cantidad de datos en bruto y que se mantienen allí hasta que sea necesario. A diferencia de un *data warehouse* jerárquico que almacena datos en ficheros o carpetas, un *data lake* utiliza una arquitectura plana para almacenar los datos.

Referencia:https://azure.microsoft.com/es-es/solutions/data-lake/

d. ***Superset***: Es una plataforma de exploración y visualización de datos, liviano, intuitivo y está repleto de opciones que facilitan a los usuarios de todos los conjuntos de habilidades la exploración y visualización de sus datos, desde gráficos de líneas simples hasta gráficos geoespaciales muy detallados.

Referencia: https://superset.apache.org/

1.12 HERRAMIENTAS PARA CONTENERIZACIÓN

a. ***Docker*:** Es una plataforma que permite crear, probar e implementar aplicaciones empaquetadas llamadas contenedores, que incluyen los componentes y librerías necesarias para que se ejecuten de manera autónoma y fiable, permitiendo el escalado horizontal.

Referencia: https://docs.docker.com/get-started/overview/

Recursos de Consulta:

○ Documentación general	https://docs.docker.com/
○ Videol de *Docker* ○ Video tutorial (5 horas)	https://bit.ly/3oqupqH https://bit.ly/2JLuhTE

b. ***Docker-compose*:** Es una herramienta para definir (componer, a través de un archivo en formato *YAML*) y ejecutar aplicaciones que ocupan varios contenedores *Docker*. Permite componer generar despliegue de aplicaciones en múltiples ambientes.

Referencia: https:/docs.docker.com/compose/
Recursos de Consulta

○ Documentación	https://docs.docker.com/compose/
○ Video tutorial (1hora)	https://bit.ly/37Ceejk

c. ***Knative (Serverless Containers Running on Kubernetes*):** Plataforma basada en *Kubernetes* para desplegar y administrar cargas de trabajo en *serverless*.

Referencia: https://knative.dev/docs

d. ***Kubernetes*:** Plataforma portable y extensible para administrar cargas de trabajo y servicios de forma automatizada a través de una configuración declarativa.

Referencia: https://kubernetes.io/es/docs/home/
Recursos de Consulta

○ What is Kubernetes (22 sessions)	https://bit.ly/37xEITg
○ Kubernetes Advance Tutorial By DevOpsSchool (25 sessions)	https://bit.ly/2JENgiR
○ Dockers and Kubernetes Tutorial for Beginners	https://bit.ly/3gehuW7
○ Complete Kubernetes Tutorial for Beginners (22 sessions)	https://bit.ly/39CQJJK

1.13 HERRAMIENTAS PARA MONITOREO (OBSERVABILIDAD)

a. ***Beats*:** Plataforma de recolección de datos open source que está conformada por una familia de productos que a través de agentes envían información operacional a una base de datos *Elasticsearch*.

Referencia: https://www.elastic.co/es/beats/

b. ***Kibana***: Aplicación de *front-end open source* que forma parte de *Elastic Stack* y permite visualización de datos y de búsqueda para los datos indexados en *Elasticsearch*.

Referencia: https://www.elastic.co/es/what-is/kibana

c. ***Prometheus*:** Conjunto de herramientas para el monitoreo y alertas que opera colectando series de datos que es almacenada en un formato de etiqueta-valor en modelo dimensional de datos.

Referencia: https://prometheus.io/

d. ***Serilog* + *Seq*:** Librería para *.Net* que permite hacer un *tracking o logging* de diagnóstico de aplicaciones de servicios y microservicios hacia archivos o a consola. *Seq* es la herramienta de visualización de la información obtenida por *Serilog.*

Referencia: https://serilog.net/

1.14 HERRAMIENTAS DE CALIDAD Y SEGURIDAD DE *SOFTWARE*

1.14.1 HERRAMIENTAS PARA ASEGURAMIENTO DE CALIDAD

a. **Herramienta de *Bugtracking*:** Es un sistema de seguimiento de errores diseñada para apoyar en la calidad de *software* y asistir a los equipos de desarrollo a realizar el seguimiento de los defectos del *software*. El *bugTracking* se realiza mediante el *Team Foundation Server*.

b. **Pruebas de Seguridad de *Software (****OWASP):* Es una metodología de pruebas de seguridad en aplicaciones *web* que tiene como objetivo determinar y combatir las causas que hacen que el *software* sea inseguro.

Referencia:https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Testing_Guide_v4.pdf

c. ***SonarQube* (**métricas del código**):** Plataforma para evaluar la calidad del código fuente, realizando un análisis estático sobre dicho código, con el objetivo de advertirnos sobre diferentes puntos a mejorar y obtener métricas que nos ayudan a mejorar nuestro código.

Referencia:
https://docs.sonarqube.org/latest/
https://docs.sonarqube.org/latest/analysis/gitlab-integration/
https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-jenkins/

1.14.2 HERRAMIENTAS PARA DIFERENCIAR COMPUTADORES DE HUMANOS

a. ***hCaptcha*:** Servicio de protección antibots, antispam y otras formas de ataques automatizados, Se integra con *Angular, Node, express*, entre otros. Es 100% compatible con la versión 2 de *Recaptcha*.

Referencia: https://www.hcaptcha.com/

b. ***Recaptcha* (versión 3 o superior):** Servicio de protección ante fraude basado en una *API* que usa un motor de análisis de riesgo que protege las aplicaciones *web*, generalmente en los formularios de *login*.

Referencia: https://www.google.com/recaptcha/about/

1.14.3 HERRAMIENTAS PARA PRUEBAS DE SEGURIDAD

Este numeral describe las principales herramientas de pruebas de seguridad. Sin embargo, de acuerdo a la naturaleza del producto de *software*, sin embargo, se podrá seleccionar herramientas complementarias.

a. ***Acunetix:*** Es un escáner de vulnerabilidades de seguridad para aplicativos y servicios *web*.

Referencia: https://www.acunetix.com/

b. ***Kali Linux*:** Es una distribución basada en Debian *GNU/Linux* diseñada principalmente para la auditoría y seguridad informática en general.

Referencia: https://www.kali.org/

c. ***Owasp ZAP*:** Es un escáner de seguridad *web* de código abierto. Pretende ser utilizado como una aplicación de seguridad y como una herramienta profesional para pruebas de penetración.

Referencia: https://owasp.org/www-project-zap/

d. ***Parrot Security OS*:** Es una distribución *GNU/Linux* basada en *Debian* con un enfoque en la seguridad informática. Está diseñado para pruebas de penetración, evaluación y análisis de vulnerabilidades, análisis forense de computadoras, navegación web anónima, y practicar criptografía.

Referencia: https://www.parrotsec.org/

1.14.4 REPOSITORIO DE CLAVES Y CERTIFICADOS DE SEGURIDAD

a. ***Keystore:*** Es una herramienta de código abierto que remplaza a la *GIU* que cuenta con utilidades de línea de comandos de *Java keytool* y *jarsigner* y presenta una interfaz gráfica de usuario más intuitiva.

Referencia: https://keystore-explorer.org/

ANEXO Nº 03

ESPECIFICACIONES Y CONSIDERACIONES TÉCNICAS PARA EL USO DE LOS REPOSITORIOS DE CÓDIGO FUENTE (v1.0)

1.1. DEFINICIONES

- ***CI/CD:*** *Continuous integration / continuous deployment* (en español Integración Continua / Desarrollo Continuo). Es un método para entregar con frecuencia un producto de *software* mediante la automatización de las etapas del desarrollo.
- ***Commit:*** Confirmación de cambio de código fuente realizado por un miembro del equipo en su *branch* local.
- ***Branch*****:** En español Rama. Es un espacio de trabajo independientes que permiten a un miembro del equipo de desarrollo trabajar sobre un mismo proyecto sin modificar los archivos estables del proyecto, brindando la facilidad del *CI/CD*.
- **ECS**: Equipo responsable de las pruebas de calidad y seguridad que está conformado por un AC y AS.
- ***Feature*****:** Es una característica particular de una aplicación, generalmente asociada con una acción. Un *feature* puede ser extraído de un caso de uso o de una historia de usuario, será utilizado por los desarrolladores para realizar el proceso de construcción y por los analistas para determinar el cumplimiento de lo solicitado por el área usuaria.
- **GIT**: Es un *software* de control de versiones de aplicaciones. Su propósito es llevar el registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.
- **Gitflow**: Es un modelo alternativo de creación de ramas en GIT en el que se utilizan ramas de función y varias ramas principales y de esta forma, dar más fluidez al proceso.
- **Github Flow**: Es una alternativa simple y ligera a GitFlow. Se basa en un flujo de trabajo basado en ramas que permite a equipos de desarrollo enfocarse principalmente en la entrega continua y está pensado para que la implementación en producción ocurra con frecuencia, incluso varias veces al día si es posible.
- **GitLab Flow**: Es una alternativa más simple a GitFlow y combina desarrollo basado en funciones y ramas de funciones con seguimiento de problemas. Incluye un conjunto de mejores prácticas y pautas para garantizar que los equipos de desarrollo de software sigan un proceso fluido para enviar funciones de forma colaborativa.
- ***Issue*****:** Es la unidad de trabajo para realizar una mejora en un sistema informático. Un *issue* puede ser el arreglo de un fallo, una característica pedida, una tarea, un pedido de documentación específico y todo tipo de solicitud al equipo de desarrollo.
- ***MR:*** *Merge request* (en español Solicitud de Fusión). Es una forma de visualizar los cambios que van a ser integrados dentro de la rama principal. Es la forma por la cual un miembro del equipo solicita que su código sea insertado en otro *branch* por lo general el *master*.
- **Principio:** Es una afirmación que no requiere justificación por sí misma, que suele hacerse evidente por la observación de fenómenos comparables y que son independientes del momento histórico, la cultura o la geografía.
- ***Pull:*** Comando *GIT* que se emplea para extraer los cambios realizados en el repositorio remoto al entorno de trabajo local.

- ***Push*:** Comando *GIT* que se emplea para realizar el envío de los cambios desde el repositorio local al repositorio remoto.
- ***Request*:** En español Petición**.** Conjunto conformado por una *URL*, un método *HTTP* y un cuerpo (opcional) para acceder a un recurso.
- ***Web services***: Es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de *software* desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los *webs services* para intercambiar datos en redes de computadoras como por ejemplo *Internet*.

1.2. ACRÓNIMOS

- **AC:** Analista de calidad.
- **AP:** Analista de plataforma.
- **AS:** Analista de seguridad.
- **DEV:** Desarrollador.
- **EA:** Equipo de analistas.
- **ED:** Equipo de desarrollo.
- **LT:** Líder técnico.

1.3. DENOMINACIONES

1.3.1. ROLES

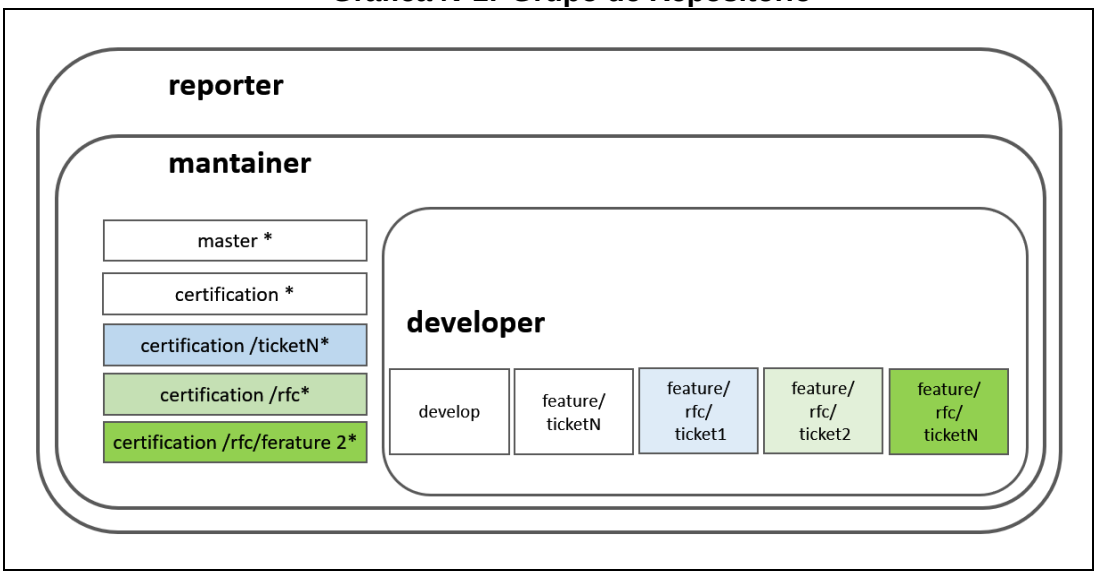
Los equipos de trabajo están compuestos por participantes de las diferentes unidades de la OTIC de la siguiente manera:

a. ***Mantainer*:** Este rol permite controlar la rama *Master* y las ramas protegidas. Para mantener el correcto mantenimiento de las ramas estables se debe contar con dos *mantainers* por proyecto:
   - Un *mantainer* del equipo responsable de la plataforma y servicios
   - Un *mantainer* del equipo responsable de la arquitectura de *software.*

b. ***Reporter*:** Este rol permite visualizar todas las ramas del repositorio, pero sin poder realizar cambios dentro de ellas, este rol se brindará al equipo responsable de las pruebas de calidad y seguridad y al gestor del equipo de proyectos para poder realizar la información remitida por los demás miembros del equipo.

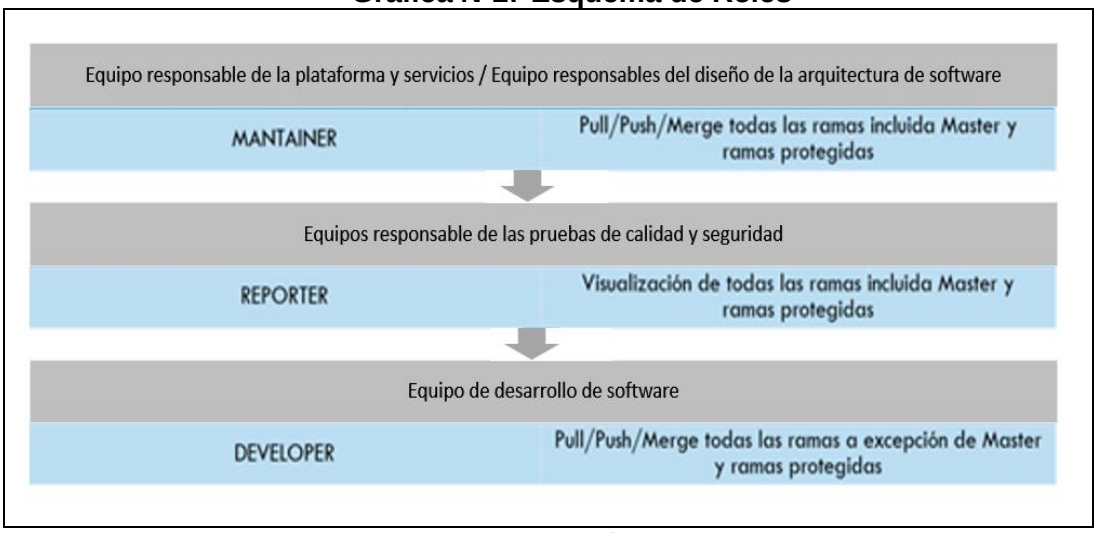
c. ***Developer*:** Este rol permite crear ramas adicionales para poder mantener el orden del repositorio al momento de hacer cambios de nuevas características o mantenimientos. Pueden solicitar *MR* a la rama *Develop* o a ramas protegidas.

Gráfica Nº1: Grupo de Repositorio



Fuente: Elaboración propia

Gráfica Nº2: Esquema de Roles



Fuente: Elaboración propia

1.3.2. GRUPOS

a. Nombre de grupo

- Los grupos serán nombrados de acuerdo con el nombre del dominio del proyecto, ejemplo: SIAGIE, APRENDO EN CASA, PERU EDUCA.
- El nombre debe ser escrito en mayúsculas.
- No debe contener caracteres especiales.
- Debe estar separado por espacios.
- Si el nombre del sistema es reconocible el nombre del grupo serán sus siglas.

b. URL de grupo

- El nombre del grupo de contener solo letras minúsculas.
- Se debe separar por guiones.

- No debe contener caracteres especiales.

1.3.3. NOMBRE DE REPOSITORIOS

Los repositorios se nombrarán de la siguiente manera:

a. El nombre se separa en 3 partes: tipo de desarrollo, nombre del proyecto en siglas y funcionamiento del aplicativo.
b. En el numeral 1.4.2 del presente anexo se muestra un diccionario de las siglas a utilizar.
c. El nombre debe ser escrito en mayúsculas.

1.3.4. RAMAS

a. Agregar un *ID* (número correlativo) de seguimiento de tareas en los nombres de las ramas.
b. Colocar un *ID* seguimiento dado que se puede realizar la trazabilidad del progreso del equipo, además se puede correlacionar el trabajo relevante de las ramas con cada tarea asignada, especialmente cuando un desarrollador está trabajando en múltiples tareas al mismo tiempo.
c. Agregar una descripción corta de la tarea relacionada con el *ID* de seguimiento, esto hace al nombre de la rama reconocible, distinguible y fácil de buscar en caso no se tenga a la mano el *ID*.
d. El descriptor debe ser conciso pero lo suficientemente descriptivo para dar una idea de qué se trata la rama.
e. Utilizar guiones como separadores.
f. En el caso de pases a calidad se debe agregar la palabra *certification* [*certification*-]1-20210920-ticket-25621

1.4. TIPOS DE REPOSITORIOS

1.4.1. REPOSITORIOS DE CONFIGURACIÓN

Este repositorio contiene los archivos de configuración necesarios para el despliegue de las aplicaciones, está limitado al rol ROL-MAN.
Existirá un repositorio por ambiente (desarrollo, certificación y producción) de esta forma los especialistas de cada unidad podrán acceder sin problemas a sus propios archivos de configuración.

Sintaxis: CONFIG-[NOMBRE-DE-SOLUCIÓN]-[AMBIENTE]

1.4.2. REPOSITORIOS DE FUENTES

Este repositorio contiene las fuentes de las aplicaciones a ser desplegadas en el MINEDU, está limitado a los roles: ROL-MAN, ROL-DEV y ROL-REP.

Sintaxis: [BACKEND|FRONTEND]-[TIPO-DE-APLICATIVO]-[NOMBRE-DESCRIPTIVO]

Tabla Nº 1: Tipos de aplicativos

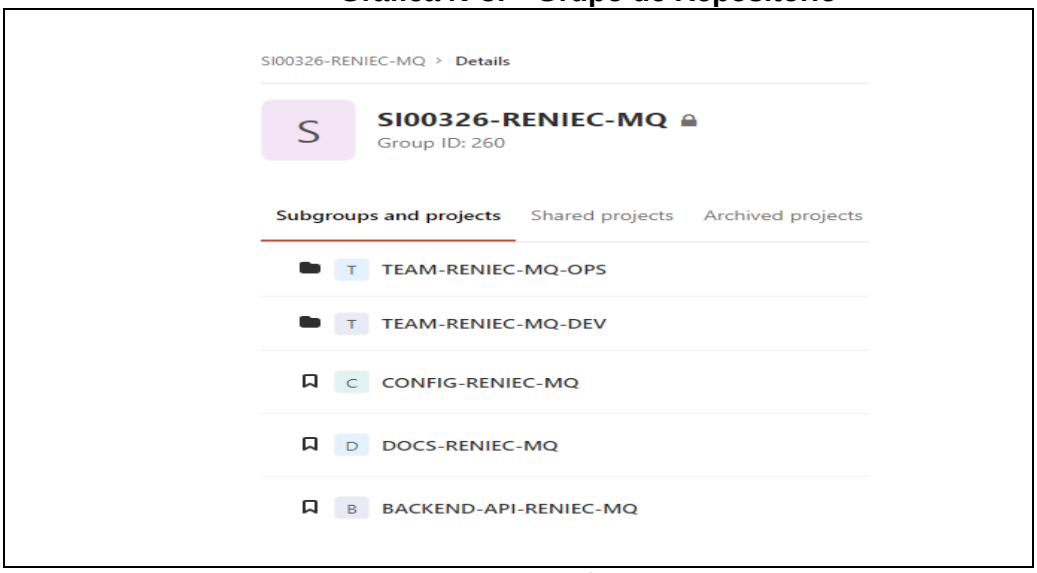
ID	Tipo aplicativo	Nombre corto
API	API	API

ID	Tipo aplicativo	Nombre corto
GA	*Gateway*	GA
WEB	Sitio *web*	WEB
WS	*Web services*	WS
MSA	Microservicio	MSA
LIB	Librerías	LIB

Fuente: Elaboración propia

Al crear los grupos, subgrupos y repositorios quedan de la siguiente manera:

Gráfica Nº3: Grupo de Repositorio



Fuente: Elaboración propia

1.4.3. REPOSITORIOS DE DOCUMENTOS

Este repositorio contiene la documentación y scripts de base de datos de las aplicaciones, está limitado a los roles: ROL-MAN, ROL-DEV y ROL-REP.

Sintaxis: DOCS-[NOMBRE-DE-SOLUCIÓN]

La siguiente gráfica muestra una estructura de carpetas que actualmente se viene utilizando, la cual puede personalizarse de acuerdo a la naturaleza del proyecto:

Gráfica Nº4: Propuesta de estructura del repositorio

	1.modelamiento-de-procesos
	2.modelamiento-de-negocio
	3.requerimiento-de-software
	4.diseño-de-interfaces
	5.matriz-de-perfiles-y-accesos

	6.especificaciones de casos-de-uso
	7.matriz-de-trazabilidad
	8.arquitectura-de-ti
	9.arquitectura-de software
	10.diseño-detallado-de-software
	11.manual-del-sistema
	12.requerimiento-tecnico
	13.software-producido

Fuente: Elaboración propia

1.5. ESTRUCTURA BASE DE REPOSITORIO

La estructura definida para la creación de repositorios se basa en mantener tres ramas esenciales que permitirán tener el orden necesario para la correcta mantenibilidad, revisión y publicación de productos de *software*.

a. **Rama *Master***: Esta rama contiene el código fuente estable que ha pasado al ambiente de producción después de haber superado el análisis de calidad y seguridad. Es una rama única y protegida solamente modificable por los *mantainer* del proyecto.

b. **Rama *Certification***: Esta rama contiene el código fuente a ser revisado por el equipo de calidad y seguridad y es la que contiene la versión estable a pasar a la rama *Master*.
   - En algunos casos se podrán requerir dos o más ramas de certificación, estas ramas auxiliares serán creadas desde la rama *Master* por única vez por un *Mantainer* o el Líder técnico del equipo de desarrollo o quien haga sus veces y se utilizará en caso de necesitar ser revisados *Tickets o RFCs* en paralelo; y una vez terminada la revisión, deben ser fusionadas con la rama Certification.
   - Estas ramas auxiliares se nombrarán de acuerdo con la siguiente sintaxis:

     auxiliar-#-aaaammdd-[exp-# | rfc-# | ticket-#]

   - El nombre debe contener la palabra auxiliar para identificar la etapa en la cual será utilizada la rama, seguida de un *ID* numérico de seguimiento, fecha de pase en el formato año, mes y día; y origen del pase: Número de expediente dentro del aplicativo Sinad, *Ticket* o *RFC*.

- auxiliar-1-20210315-rfc-125-2020
- auxiliar-3-20210830-ticket-0001523
- auxiliar15-20210725-exp-138-usi

- El LT solicitará un MR desde estas ramas auxiliares hacia la rama protegida Certification, que no será aceptado por el equipo de Plataforma mientras el equipo de Calidad y Seguridad no brinde la conformidad de las pruebas y se solicite el pase a producción.

c. Rama Development: Esta rama contiene el código fuente en proceso de desarrollo, que si bien no se considerada estable, se recomienda que sea lo más estable posible, puesto que a partir de ella el equipo de desarrollo generará ramas auxiliares para construir o modificar features solicitados. En esta rama, el equipo de desarrollo puede solicitar la fusión de sus cambios para que el líder técnico del equipo de desarrollo o quien haga sus veces realice la integración y se valide el correcto funcionamiento del producto o componente de software a ser revisado por el equipo responsable de las pruebas de calidad y seguridad**.**

La herramienta *GIT* permite el trabajo colaborativo, en tal sentido, se pueden generar múltiples ramas auxiliares para desarrollar cada *feature* solicitado, se debe tomar en cuenta:

- Una rama debe cumplir con el principio de responsabilidad única.
- Una rama no debe durar demasiado tiempo abierta, lo recomendable es de dos a cinco días máximo.
- Una rama debe estar nombrada con letras minúsculas, números y guiones solamente [a-z0-9-]. No debe contener caracteres especiales, incluso “ñ”.
- De preferencia, se debe nombrar la rama en inglés.
- No se deben dejar ramas huérfanas. Si se ha concluido el desarrollo se deben fusionar con la rama *Development*, y si por alguna razón no se ha culminado con las tareas de desarrollo deben eliminarse hasta que se retome nuevamente.

1.6. FLUJOS DE TRABAJO

Para facilitar la adopción progresiva de una cultura *DEVSECOP*S, a continuación, se describe el flujo de trabajo con la herramienta de versionado *GIT*, mecanismo de integración de despliegue automatizado y continuo que permite el trabajo colaborativo de programación de forma homogénea y productiva, tomando las mejores prácticas y adaptándolas al desarrollo propio.

1.6.1. FLUJO DE TRABAJO NORMAL

a. El DEV inicia sus actividades trabajando en base al *GitFlow* con ramas *Feature* en las cuales se desarrollan las diferentes tareas.

b. Una vez terminada la tarea, el DEV realiza una MR a la rama *Development*, el LT o quién haga sus veces en el equipo de desarrollo, revisa y acepta o niega la solicitud.

c. En el ambiente de desarrollo, el LT realiza el despliegue automatizado con la herramienta de despliegue *Jenkins*, pasando por un proceso de control de calidad y seguridad automatizado.

d. En el ambiente desarrollo, el EA realiza pruebas funcionales, integración, entre otras; y de encontrar alguna observación lo informa al LT y equipo de desarrollo. El LT solicita al DEV que proceda a implementar las correcciones para luego realizar una nueva revisión; en caso contrario, el LT realiza el pase a calidad solicitando un MR de la rama *Development* a la rama *Certification*.

e. El AC verifica que los objetos del pase sean los correctos y solicita al AP o quien haga sus veces el despliegue automatizado con la herramienta de despliegue *Jenkins*, pasando por un proceso de control de calidad y seguridad automatizado.

f. En el ambiente de certificación, el ECS ejecuta las pruebas necesarias.

g. Concluidas estas pruebas y de encontrarse alguna observación, el AC registra las observaciones en la herramienta de seguimiento de observaciones y envía los informes de Calidad y Seguridad.

h. El ED recibe los informes de Calidad y Seguridad, de existir observaciones procede a realizar los cambios y mejoras para iniciar nuevamente un pase a calidad. De no existir observaciones y al contar con la aprobación del ECS, el LT o quien haga sus veces en el ED solicita al AP o quien haga sus veces que realice el pase a producción; el AP realiza un MR de *Certification* a *Master*.

1.6.2. FLUJO DE TRABAJO CON MÚLTIPLES REVISIONES

a. El DEV inicia sus actividades trabajando en base al *GitFlow* con ramas *Feature* en las cuales se desarrollan las diferentes tareas.

b. Una vez terminada la tarea, el DEV solicita al LT o quien haga sus veces en el ED que revise y acepte lo desarrollado.

c. En el ambiente de desarrollo, el LT realiza el despliegue automatizado con la herramienta de despliegue *Jenkins* pasando por un proceso de control de calidad y seguridad automatizado.

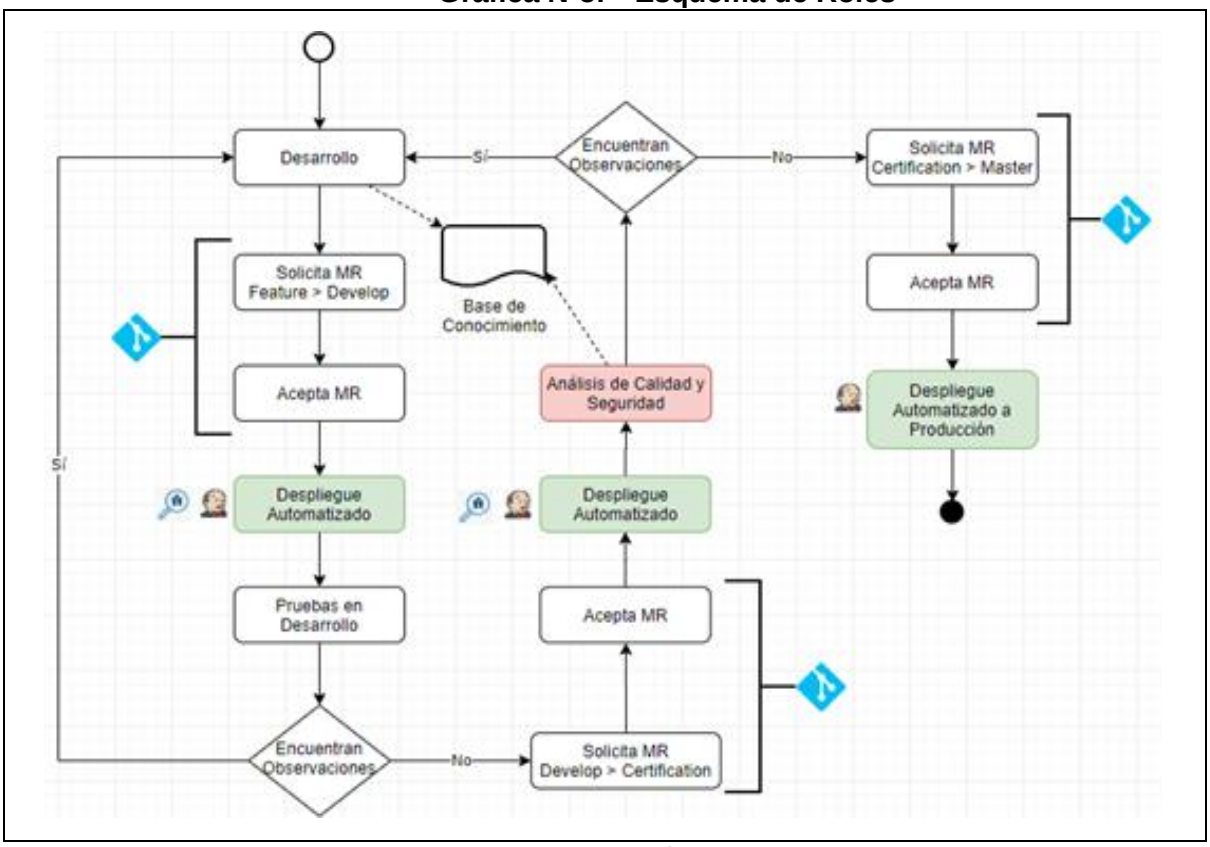
d. En el ambiente desarrollo, el EA realiza pruebas y de encontrarse alguna observación lo informa al LT y ED. El LT solicita al DEV que proceda a implementar las correcciones para luego realizar una nueva revisión; en caso contrario, de no encontrar observaciones, el LT realiza el pase a calidad solicitando un MR de la rama de desarrollo utilizada a una rama auxiliar de *Certification* que es creada por el LT del ED o quien haga sus veces a partir de la rama *Master* del repositorio, siendo esta rama de certificación bloqueada automáticamente por el sistema para evitar la inclusión de objetos al pase después de la revisión de calidad.

e. El AC verifica que los objetos del pase sean los correctos y solicita al AP o quien haga sus veces realice el despliegue automatizado con la herramienta de despliegue *Jenkins* al ambiente de certificación pasando por un proceso de control de calidad y seguridad automatizado.

f. En el ambiente de certificación, el ECS realiza las pruebas manuales y automatizadas necesarias.

g. Concluidas estas pruebas y de encontrarse alguna observación, el AC remite el Informe de Calidad y Seguridad con las observaciones e indica al ED proceder a realizar los cambios y mejoras para iniciar nuevamente un pase a Calidad. En caso de no existir observaciones y de contar con la aprobación del ECS, el LT o quien haga sus veces en el ED solicita al AP o quien haga sus veces que realice el pase a producción a través de:
   - La aceptación del *MR* de la rama de desarrollo a la rama auxiliar de calidad.
   - Generar un *MR* de la rama auxiliar de calidad a la rama *Certification.*
   - Eliminar la rama auxiliar de calidad.
   - Generar un *MR* de *Certification* a *Master*.

Gráfica Nº5: Esquema de Roles



Fuente: Elaboración propia

ANEXO Nº 04

ESPECIFICACIONES Y CONSIDERACIONES TÉCNICAS PARA EL DISEÑO Y USO DE LAS BASES DE DATOS *SQL Y NOSQL* (v1.0)

1.1 DEFINICIONES

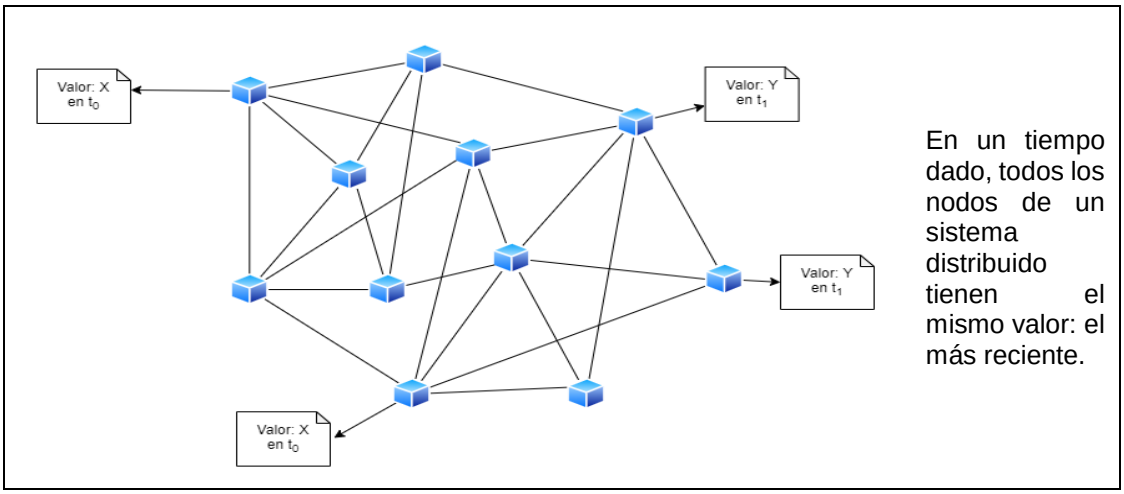
- **Consistencia eventual:** Es un modelo de consistencia de datos que, en oposición a la consistencia inmediata, todos los observadores del sistema podrán ver los cambios realizados en los repositorios de datos en intervalos de tiempo diferentes. La consistencia eventual se acompaña de una política de resolución de conflictos, por lo que existe una garantía futura de la consistencia.
- ***DDD:*** *Domain Driven Design* (en español Diseño Guiado por el Dominio). Es un enfoque o técnica para el desarrollo de *software* que utiliza patrones y recomendaciones orientadas a diseñar un modelo de dominio o lógica del negocio que ayude a tratar la complejidad de las aplicaciones.
- ***Entity Framework*:** Es una biblioteca que facilita el acceso a datos a los desarrolladores. Así, al acceder a datos, en lugar de utilizar un lenguaje específico se tiene un *ORM* que habilita paradigmas habituales de la programación orientada a objetos: clases y objetos. En lugar de trabajar sobre tablas y relaciones, se trabaja con objetos y propiedades.
- ***ETL:*** *Extract, Transform and Load* (En español Extraer, Transformar y Cargar). Es un tipo de integración de datos que mezcla datos de múltiples fuentes.
- ***Framework*:** En español Marco de Trabajo. Es un conjunto de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.
- **Modelo de datos con referencias *NOSQL*:** Es un modelo normalizado y mantiene referencias entre dos documentos para indicar una relación entre sí.
- ***NOSQL*:** Significa "no solo *SQL*". Agrupa a un conjunto de sistemas de gestión de bases de datos que difieren del modelo Sistema de Gestión de Bases de Datos Relacionales ya que no usan *SQ*L como lenguaje principal de consultas. Los datos almacenados no requieren estructuras fijas como tablas, tampoco soportan operaciones *JOIN*. No garantizan completamente ACID (atomicidad, consistencia, aislamiento y durabilidad).
- **Principio:** Es una afirmación que no requiere justificación por sí misma, que suele hacerse evidente por la observación de fenómenos comparables y que son independientes del momento histórico, la cultura o la geografía.
- **Recurso:** En el marco de los sistemas de información distribuidos, es cualquier elemento de información distinguible y disponible en internet. Por ejemplo, un documento electrónico, una imagen, un servicio, etc.
- ***Response*:** En español Respuesta. Conjunto de datos que son emitidos por un servidor ante una petición y que está conformado por un encabezado y un cuerpo. El encabezado está conformado por un código de estado y parámetros.
- **Sistema distribuido:** Es un conjunto de servidores que trabajan de forma coordinada, a través del intercambio de mensajes, para conseguir un objetivo. En dicho sistema, el estado y los programas se guardan en múltiples servidores. Si bien los procesos que tienen lugar están separados entre los diferentes participantes, para el usuario parece que está trabajando con un único servidor.
- **Teorema de CAP:** Conocido también como Teorema de Brewer que afirma que no es posible para un sistema de cómputo distribuido garantizar simultáneamente más de dos características de las tres siguientes: Consistencia, Disponibilidad y Tolerancia a la partición.

1.2 TEOREMA DE CAP PARA SISTEMAS DISTRIBUIDOS

El análisis por este teorema permite valorar la configuración y comportamiento esperado del *software* por desarrollar a fin de sustentar técnicamente la selección de un tipo de base de datos. El proceso se inicia con la identificación de las características que se priorizarán en la construcción del sistema distribuido, que es una tarea asociada a la identificación de requerimientos no funcionales. Estas características son:

A. Consistencia (*Consistency*): Garantiza que una lectura retornará siempre la actualización más reciente de un registro particular. Siempre que se modifique un dato, el cambio debe reflejarse en todos los nodos de la base de datos.

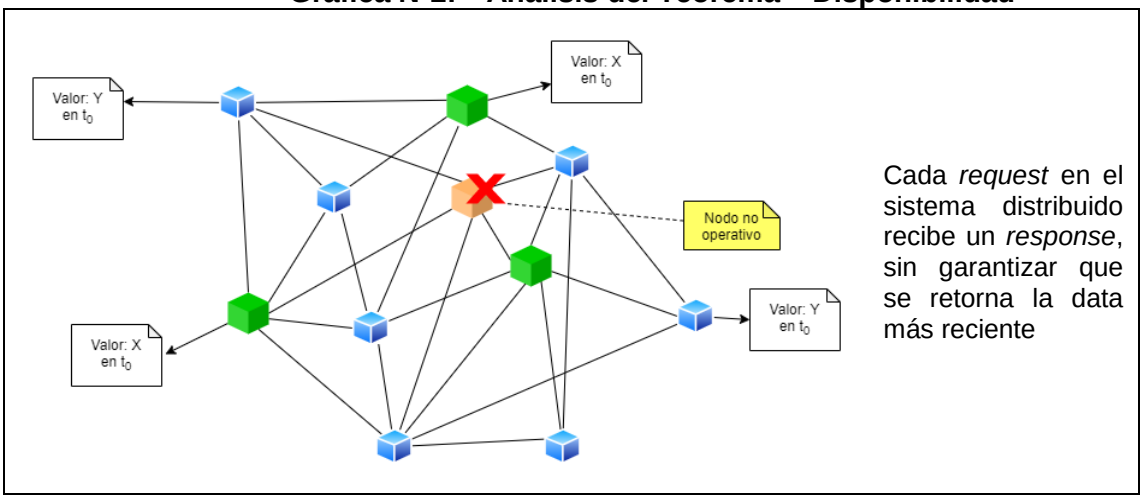
Gráfica Nº1: Análisis del Teorema – Consistencia



Fuente: Elaboración propia

B. Disponibilidad (*Availability*): En los sistemas distribuidos, los clientes pueden leer y escribir, aunque uno de los nodos haya dejado de funcionar, sin embargo, no es posible garantizar que las respuestas contengan la última escritura.

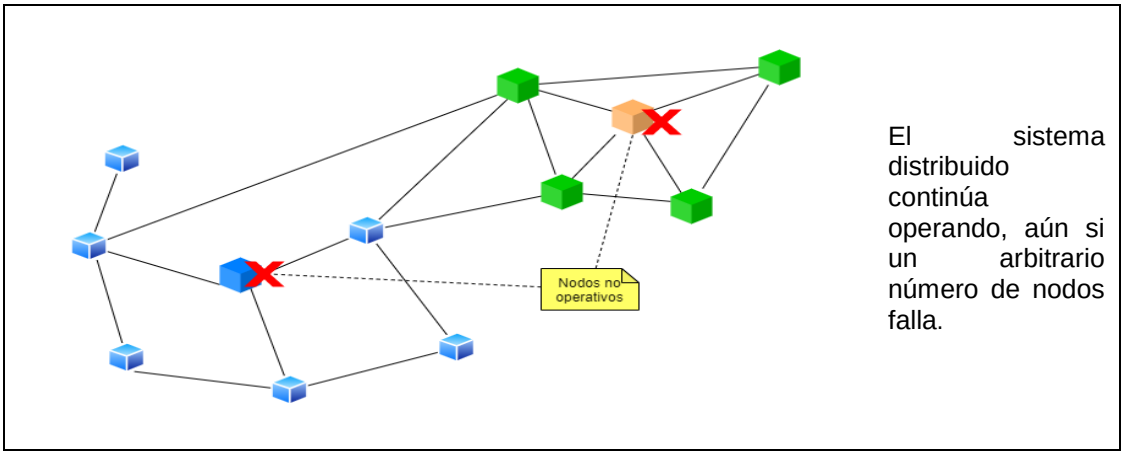
Gráfica Nº2: Análisis del Teorema – Disponibilidad



Fuente: Elaboración propia

C. Tolerancia al Particionamiento (*Partition Tolerance*): En algunas ocasiones, los sistemas distribuidos están divididos en particiones y es posible que existan problemas de comunicación entre ellas, por lo cual el sistema debe seguir funcionando. El sistema nos debe seguir funcionando, aunque algunos nodos no se encuentren disponibles ya que la información es consistente en todos los nodos.

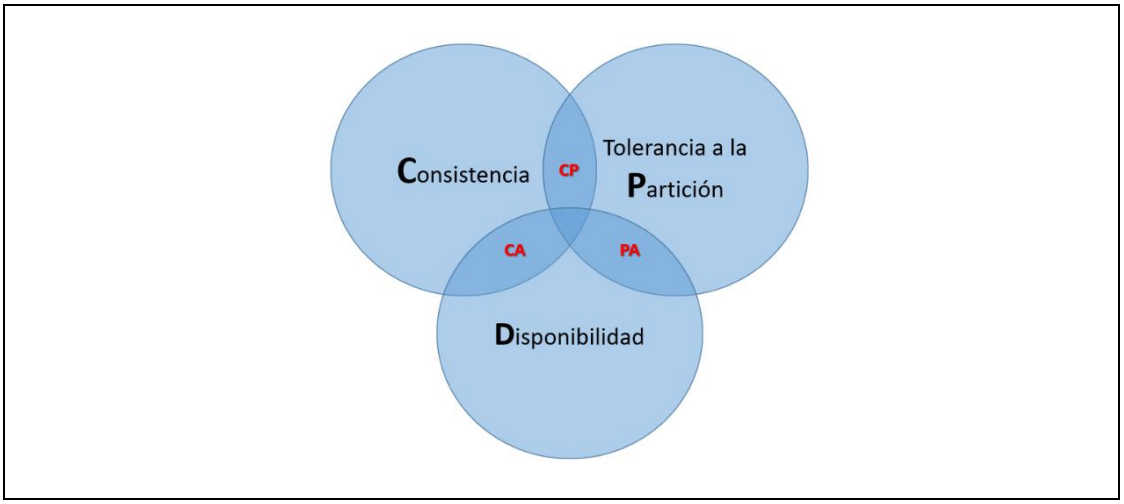
Gráfica Nº3: Análisis del Teorema – Tolerancia al Particionamiento



Fuente: Elaboración propia

Si se intersecan las características requeridas para las bases de datos, se generan las siguientes combinaciones:

Gráfica Nº4: Análisis del Teorema CAP – Intersecciones



Fuente: Elaboración propia

Análisis de las combinaciones:

- **Combinación CA**: garantizan consistencia y disponibilidad, pero tienen problemas con la tolerancia a particiones. Este problema lo suelen gestionar replicando los datos. Entre los tipos de aplicaciones que se encuentran en este grupo se tiene:
  - Aplicaciones que implican toma de decisiones con respecto a montos: sistemas de presupuesto, *datawarehouses, datamarts*.
  - Aplicaciones en las que los requerimientos y las entidades de negocio están claramente establecidos y están calificadas como datos sensibles.
  - Aplicaciones fácilmente escalables verticalmente.

- Aplicaciones que utilizan información proveniente de otros sistemas, ya sea por *API* o por *ETL.*
- Bases de datos: *SQL Server y MySQL.*

- **Combinación AP**: garantizan disponibilidad y tolerancia a particiones, pero no la consistencia, al menos de forma total. Algunas de ellas consiguen una consistencia parcial a través de la replicación y la verificación. Entre los tipos de aplicaciones que se encuentran en este grupo se tiene:
  - Aplicaciones que han previsto un crecimiento rápido de usuarios concurrentes.
  - Aplicaciones altamente distribuidas.
  - Aplicaciones que gestionan las preferencias de los usuarios, sistemas de gamificación con asignación de puntajes inmediatos.
  - Bases de datos: *Cassandra y CouchDB*.

- **Combinación CP**: garantizan consistencia y tolerancia a particiones. Para lograr la consistencia y replicar los datos a través de los nodos, sacrifican la disponibilidad. Entre los tipos de aplicaciones que se encuentran en este grupo se tiene:
  - Aplicaciones que manejan grandes volúmenes de información en periodos muy cortos de tiempo.
  - La información no es sensible y los maestros y transacciones tienen pocos atributos (no más de 20).
  - Aplicaciones en las que los requerimientos y las entidades de negocio no están claramente establecidos.
  - Aplicaciones de mensajería instantánea, analítica, sistemas que hacen uso de la sesión del usuario.
  - Bases de datos: *MongoDB*.

Se debe tener presente que esta clasificación no es rígida, ya que algunos gestores de base de datos pueden configurarse para cambiar su comportamiento[8].

1.3 DIFERENCIAS ENTRE EL TIPO DE BASE DE DATOS

Para la selección del tipo[9] de base de datos se debe evaluar las siguientes diferencias:

Tabla Nº 1: Diferencias entre Tipos de Base de Datos

CARACTERÍSTICA[10]	*SQL*	*NOSQL*
Almacenamiento de datos	Almacena información en forma de tablas. Cada fila contiene información de una entidad específica.	Los datos son almacenados en diferentes formatos que pueden incluir documentos, pares clave-valor, grafos, entre otros.
Esquemas	La creación de tablas se basa en esquemas e implica una tarea compleja alterar el esquema una vez creado.	El esquema es altamente dinámico y la información puede ser cambiada fácilmente.

[8] Por ejemplo, MongoDB es CP por defecto. Pero también podemos configurar el nivel de consistencia, eligiendo el número de nodos a los que se replicarán los datos. O podemos configurar si se pueden leer datos de los nodos secundarios (en MongoDB solo hay un servidor principal, que es el único que acepta inserciones o modificaciones). Si permitimos leer de un nodo secundario, se sacrifica consistencia en favor de la disponibilidad.
[9] https://infonomics-society.org/wp-content/uploads/ijds/published-papers/volume-8-2017/Security-Vulnerabilities-of-NoSQL-and-SQL-Databases-for-MOOC-Applications.pdf
[10] Una característica o ventaja en un tipo de base de datos, se corresponde con una desventaja del otro tipo, y viceversa.

CARACTERÍSTICA[10]	*SQL*	*NOSQL*
Escalabilidad	La escalabilidad es vertical. Es posible escalar bases de datos a través de múltiples servidores, pero se convierte en un proceso costoso.	Es escalable horizontalmente de manera natural.
Integridad	Cumplen los principios ACID.	No cumplen los principios ACID. Aplican consistencia eventual.

Fuente: Elaboración propia

1.4 CONSIDERACIONES PARA EL DISEÑO LÓGICO DE LA BASE DE DATOS

1.4.1 MODELO CONCEPTUAL

a. El modelo conceptual es independiente de la implementación física de la base de datos.
b. No se debe incluir los atributos de auditoría en los modelos conceptuales.
c. Se debe incluir todas las entidades identificadas independientemente de la forma en la que se implementará físicamente.
d. Se debe evitar incluir en los modelos, entidades tales como “Catálogo” (o “Enumerado”) en donde se unen diferentes entidades, especialmente referidas a tipos o estados (que únicamente tienen como dos atributos: código y descripción).
e. En el modelo conceptual, es posible la inclusión de fuentes de datos externas tales como una *API*, en especial cuando se está modelando la base de datos de una aplicación orientada a servicios o microservicios.
f. En el caso de que se requiera redundar datos a fin de desacoplar los modelos, será necesario que el nombre los campos tengan el prefijo “redundante”.

1.4.2 NOMBRE DE LAS ENTIDADES PARA BASES DE DATOS *SQL*

a. Para los casos de entidades maestras, el nombre de la entidad debe consignarse en singular.
b. Se debe evitar los nombres de más de dos palabras y evite las preposiciones y artículos dentro de los nombres de las entidades, así como también las abreviaturas.

Tabla Nº 2: Ejemplo de Nombres

CORRECTO	INCORRECTO	OBSERVACIÓN
Tipo Documento	Tipo de Documento	Es incorrecto por tener más de dos palabras. Se ha suprimido la preposición.
Estudiante	estudiante	Es incorrecto por empezar con minúscula.
Contenido	Contenidos	Es incorrecto por estar en plural.
Institución Educativa	Inst_Educ	Es incorrecto por ser una abreviatura

Fuente: Elaboración propia

c. Las entidades hijo que dependen de un padre se deben nombrar de la siguiente manera:
   - El nombre de la entidad hija va en plural.

- Luego colocar la palabra “por”.
- Finalmente, consignar el nombre de la entidad padre.

Tabla Nº 3: Ejemplo de Nombres de Entidades

ENTIDAD PADRE	ENTIDAD HIJO	ENTIDAD RELACIÓN
institución educativa	servidor publico	servidores públicos por institución educativa
servidor publico	familiar	familiares por servidor publico

Fuente: Elaboración propia

Nota: Generalmente estas entidades resuelven las relaciones de muchos-a-muchos.

d. No se debe permitir que la herramienta de modelamiento resuelva las relaciones de muchos-a-muchos al momento de generar el modelo físico, para evitar que la relación la genere la herramienta en lugar de poder seleccionar la propia.

e. El estereotipo debe usarse únicamente en el desarrollo de bases de datos monolíticas.

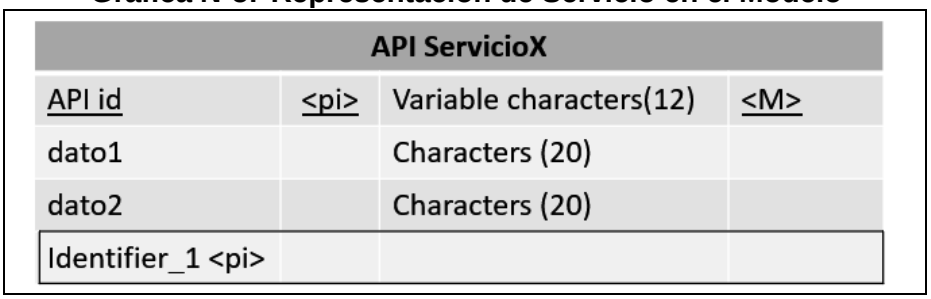
Tabla Nº 4: Ejemplo de Estereotipos

NOMBRE DE ENTIDAD	ESTEREOTIPO	GRAFICO
Servidor publico	maestro	<<maestro>> servidor publico
Motivo accion	sistema	<<sistema>> motivo accion

Fuente: Elaboración propia

f. Las entidades pueden representar servicios externos, siempre que el diseñador de bases de datos desee representar a los servicios como fuente de datos. En este caso, el nombre de la entidad debe iniciar con el prefijo “*API_*”. Como atributos, el diseñador podrá incluir a los datos que son relevantes para la base de datos. Esta entidad no debe generar ninguna tabla física en la base de datos. En general, este tipo de entidades se representan así:

Gráfica Nº5: Representación de Servicio en el Modelo



Fuente: Elaboración propia

1.4.3 NOMBRE DE LOS ATRIBUTOS DE ENTIDADES PARA BASES DE DATOS *SQL*

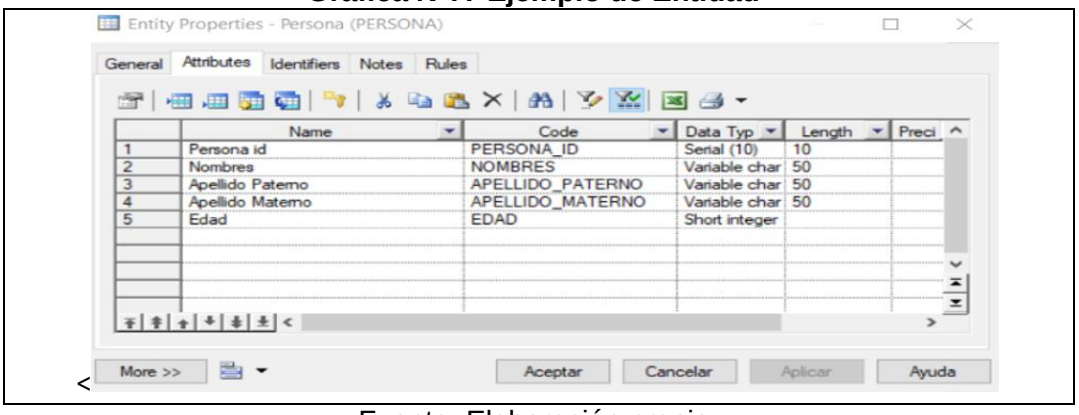
a. Se debe tener un nombre descriptivo. Opcionalmente, pueden tener como prefijo el nombre de la entidad, pero si se adopta este modelo, debe cumplirse para toda la base de datos.

Gráfica Nº6: Ejemplo de Nombres de Atributos

PERSONA			
Persona id	<pi>	Serial (10)	<M>
Nombres		Variable characters (50)	
Apellido Paterno		Variable characters (50)	
Edad		Variable characters (50)	
Identifier_1 <pi>		Short integer	

Fuente: Elaboración propia.

Gráfica Nº7: Ejemplo de Entidad



Fuente: Elaboración propia.

b. Si un atributo forma parte de la clave primaria debe estar conformado por el nombre de la entidad seguido por los caracteres "*id*" (Identificador).
c. Los atributos de estado cobran importancia en el esquema de los patrones de arquitectura de microservicios y en la gestión de eventos. Por ello, las descripciones de los estados deben almacenarse en las tablas denominadas "enumerado".
d. Se debe utilizar en lo posible un conjunto de dominios para uniformizar los tipos de dato de los atributos de las entidades. A continuación, se detallan algunos ejemplos de dominios utilizados:

Tabla Nº 5: Ejemplos de Dominios

NOMBRE DEL DOMINIO	TIPO DE DATO	DESCRIPCIÓN DEL DOMINIO
Id	Numeric	Identificador de entidad. Es autoincremental (serial).
Abreviatura	Variable multibyte (10)	Domino de los valores abreviados o siglas de un registro.
Código	Variable characters (20)	Domino de los códigos de un registro.

NOMBRE DEL DOMINIO	TIPO DE DATO	DESCRIPCIÓN DEL DOMINIO
Descripción corta	Variable multibyte (30)	Domino de los valores descripciones o nombres de máximo 30 caracteres de un registro.
Descripción larga	Variable characters (255)	Domino de los valores descripciones o nombres de máximo 255 caracteres de un registro.
Dinero	Money (10,2)	Domino de los valores de moneda o dinero.
Fecha	Date	Domino de los valores de solo fecha de un registro.
Identificador	Integer	Domino de los identificadores autogenerados e incrementales para un registro.
Identificador detalle	Integer	Domino de los identificadores autogenerado incrementales por cada llave de la(s) tabla(s) dependiente(s).
Texto	Variable multibyte	Domino de los valores de textos largo o comentarios de más de una línea de un registro.
Enumerado	Integer	Dominio de los valores de los estados y valores finitos.

Fuente: Elaboración propia.

1.4.4 DEFINICIONES DE ENTIDADES Y LOS ATRIBUTOS

a. Las definiciones de las entidades y atributos, es decir el texto explicativo de la entidad o de los atributos que se incluye en el diccionario de datos, deben hacerse en un lenguaje, claro y sencillo para que puedan ser entendidas por los analistas y desarrolladores.
b. Se debe evitar las definiciones con circularidad o tautologías (Por ejemplo: evitar definir a la entidad “Permiso” como: “es el permiso solicitado”) También evitar hacer definiciones con sinónimos únicamente.
c. Se debe evitar las definiciones extensas y/o demasiado precisas.

1.5 CONSIDERACIONES PARA EL DISEÑO FÍSICO DE LA BASE DE DATOS

1.5.1 DISEÑO FÍSICO DE BASES DE DATOS *SQL*

a. El modelo físico de la base de datos, en ocasiones puede generarse a partir del modelo conceptual o del modelo lógico. En tal caso, la herramienta de modelamiento será la que determine los nombres, relaciones y demás objetos de la base de datos.
b. A partir del modelo físico de la base de datos, se obtiene el diccionario de datos, el cual debe formar parte del documento de diseño detallado bajo el formato vigente.
c. En la generación de la base de datos a partir del modelo físico, se debe seguir las siguientes consideraciones en el modelamiento:

- En la medida que la generación de la base de datos se realiza desde la herramienta de modelamiento, ésta debe ser configurada para que la generación de objetos contenga los siguientes estilos de nomenclatura.

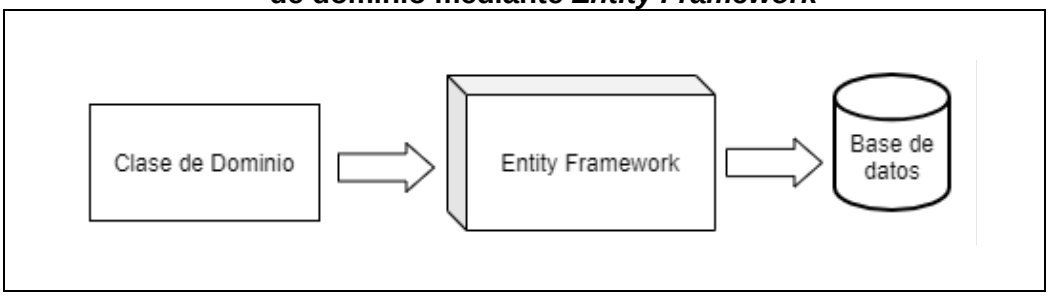
Tabla Nº 6: Nombre de objetos en Bases de Datos

OBJETO	ESTILO	EJEMPLOS
Nombre de la tabla	lower_snake_case	persona, tipo_resolucion
Nombre de campos	UPPER_SNAKE_CASE	PERSONA_ID NOMBRES APELLIDO_PATERNO SITUACION JUSTIFICACION

Fuente: Elaboración propia.

- Todas las sentencias *SQL* que se elaboren deben respetar estrictamente la sensibilidad de mayúsculas y minúsculas presente en la base de datos.
- Los *constraints* de integridad referencial generados desde el modelo conceptual no deberían ser eliminados ni modificados.
- Los modelos físicos obtenidos a partir de la aplicación de ingeniería reversa sobre la base de datos física, permitirán generar los modelos lógicos o conceptuales.
- Una de las formas de obtener la base de datos física es la aplicación del enfoque *Code-First*, que está disponible desde *Entity Framework 4.1* y es usado con el diseño orientado a dominios (*Domain Driven Design - DDD*). Este mecanismo puede utilizarse cuando se trata de una base de datos pequeña. Una base de datos pequeña, se puede considerar como tal, si es que tiene menos de diez entidades/tablas. Por ejemplo, el siguiente diagrama esquematiza la generación de una base de datos a partir de una clase de dominio.

Gráfica Nº8: Ejemplo de generación de base de datos a partir de una clase de dominio mediante *Entity Framework*



Fuente: Elaboración Propia.

1.5.2 DISEÑO FÍSICO DE BASES DE DATOS *NOSQL*

a. Se debe generar los objetos directamente sobre una base de datos.
b. Para el caso de *MongoDB*, que maneja documentos en formato *JSON*, es útil usar herramientas que permitan generar objetos en dicho formato.

1.5.3 MAPEADORES

Para facilitar el acceso a las estructuras de datos sin necesidad de utilizar sentencias (de consultas o de actualización) escritas directamente en lenguaje propio de la base de datos se debe desarrollar componentes que interactúan con la capa de datos, llamados mapeadores. El uso de los mapeadores está permitido en los siguientes casos:

- Cuando el tiempo de respuesta NO es un factor crítico del servicio, módulo o subsistema a desarrollar.
- Cuando el modelo de datos no es complejo.
- Cuando se cubre el 95% de entidades del modelo conceptual.

Tipos de mapeadores:

- Para bases de datos *SQL*: *ORM* (*Object Relational Mappings*).
- Para bases de datos *NoSQL*: *ODM* (Object Document *Mappings*).

Para implementar mapeadores se puede utilizar:

- ORM: *Entity Framework Core, Dapper*.
- ODM: *Mongoose.*

1.6 CONSIDERACIONES PARA BASES DE DATOS SQL

1.6.1 CONSIDERACIONES GENERALES

a. El tipo de *collation* de la base de datos debe ser *Latin1_General_CS_AS*. De requerirse un *collation* diferente, debe ser indicado en el documento de arquitectura de cada proyecto. De la misma forma con el formato de codificación de caracteres que, por defecto es *UTF*-8.
b. Tener en cuenta la recomendación de *Microsoft* de la NO utilización de palabras reservadas de *MS-SQL*, bajo responsabilidad.
c. Por defecto, no se aceptan espacios en blanco en medio de los identificadores de los objetos de la base de datos.
d. En base a la normativa[11] actual la habilitación de los registros (*log*) de auditoría de las bases de datos se hará a demanda para las diferentes acciones (*Insert, Update, Delete, Select, Drop, etc*.), el actor (usuario), la fecha-hora, *IP* desde donde se generó la operación y los campos de la tabla afectada. Asimismo, se debe registrar en la tabla afectada, siempre y cuando así lo requiera el modelo de datos: el usuario de creación, fecha-hora de creación, actor de modificación, fecha-hora de modificación, *IP* del equipo desde donde se hizo la actualización.
e. Los usuarios que son creados en las diferentes bases de datos para las conexiones a las mismas deben iniciar con el mínimo privilegio y según se indique en los documentos de análisis se habiliten los permisos correspondientes a los roles y funciones que realicen dichos usuarios.
f. La instancia contenedora de las bases de datos, así como las propias bases de datos deben habilitar sus módulos de seguridad para mitigar riesgos como denegación de servicio, inyección de código, entre otros.
g. Usar la metodología de seguridad aplicada a la gestión de bases de datos, para garantizar su resiliencia ante desastres, su fortaleza ante ataques contra la

[11] Normativa vigente: Ley de protección de datos personales LEY Nº 29733, Norma Internacional de Seguridad de la Información – ISO27001, Directiva de copia de base de datos entre otros.

seguridad, entre otros, para este punto se recomienda hacer uso de las buenas prácticas indicadas en *OWASP: Database Security Cheat Sheet*.

h. Los campos cifrados y ofuscados deben ser elegidos aplicando el marco de la Ley 29733, el criterio de clasificación de la información del MINEDU y el criterio de clasificación de la información que indique el usuario o dueño de la base de datos.

i. Todo procedimiento almacenado debe verificar que sus variables de entrada se encuentran sanitizadas, es decir no deben contener caracteres o expresiones que permitan la inyección de código.

j. Se debe cumplir con la siguiente nomenclatura en bases de datos *SQL*:

Tabla Nº 7: Nomenclatura de Bases de Datos

NOMBRE	Prefijo: db_ Nombre: <Nombre del sistema o servicio>
DESCRIPCIÓN	<Nombre del sistema abreviado> es un nombre descriptivo o representativo que permita determinar fácilmente su propósito, escrito en minúscula, sin caracteres especiales. En caso de que se trate de un nombre compuesto, se puede concatenar con el carácter underscore o subguión: "_"
EJEMPLOS	db_sinad db_contabilidad db_alerta_sinad db_asignador_expediente

Fuente: Elaboración propia.

Tabla Nº 8: Nomenclatura de Esquemas

NOMBRE	<Nombre del esquema>
DESCRIPCIÓN	Debe ser un nombre descriptivo o representativo con respecto al estereotipo definido en el modelo conceptual o al microservicio en la aplicación del patrón "Microservicio por esquema". Debe estar escrito en minúscula y sin caracteres especiales y con longitud no mayor a 30 caracteres.
EJEMPLOS	recibos pecosas matriculas
ADICIONALES	Existen los siguientes tipos de esquemas **Esquema** — **Descripción** Transaccional — Contiene a las tablas transaccionales persiguen dos propósitos: el acumular datos de eventos en el momento que ocurran y actualizar archivos maestros para reflejar los resultados de las transacciones. Guardan información sobre los eventos y sobre los cuales se calculan datos. Maestro — Contiene a las tablas maestras almacenan datos de las entidades de negocio de la organización. Histórico — Contiene a las tablas históricas reflejan todos los cambios de valor

	Paramétrica	que afectaron a una entidad determinada. Contiene a las tablas paramétricas contienen datos de reglas o variables que afectan el comportamiento de todo el sistema o servicio.
	Sistema o servicio	Las tablas de sistema o servicio contienen datos de los valores que forman parte de los requerimientos no funcionales del sistema o servicio.
	Auditoría	Contiene a las tablas de auditoría donde se encuentra la información de los valores históricos de eventos. Puede referirse a operaciones *CRUD*, accesos a servicios o sistemas, mensajes de error.
	Microservicio	Es el nombre del microservicio que está implementando.

Fuente: Elaboración propia.

Tabla Nº 9: Nomenclatura de Tablas

NOMBRE	<Nombre de esquema>. <Nombre de tabla >
DESCRIPCIÓN	• Debe ser un nombre descriptivo, escrito en singular y con minúsculas, sin caracteres especiales y no tiene límites de caracteres establecidos, sin embargo, se recomienda que no sean más de 30 caracteres, a excepción de los límites establecidos por el manejador de base de datos. • En caso de tener un nombre compuesto, se deberá unir con el caracter underscore “_”. • Si se trata de una tabla hija de una relación “uno a muchos” el nombre de la tabla lleva el nombre del padre y seguido por el conector “_por_” y el nombre de la tabla en plural.
EJEMPLOS	maestro.servidor_publico maestro.familiares_por_servidor_publico transaccional.accion historico.historial_por_plaza msa_matricula.inscripcion

Fuente: Elaboración propia.

Tabla Nº 10: Nomenclatura de Columnas

NOMBRE	<Nombre de columna>
DESCRIPCIÓN	• Debe ser un nombre descriptivo, escrito en singular (a menos que almacene un arreglo) y en MAYÚSCULA, sin caracteres especiales y no tiene un límite de caracteres establecido (recomendable no mayor de 30 caracteres), a excepción de los límites establecidos por el manejador de base de datos.

	• En caso de tener un nombre compuesto, se debe unir con el caracter *underscore*: "_". • Las columnas que son PK deben contener el nombre de la tabla seguido del sufijo:"_ID"m de esta forma cuando sean referenciadas como FK se podrá reconocer rápidamente.
EJEMPLOS	PERSONA_ID int NOMBRES nvarchar(50) CORREO_ELECTRONIC O nvarchar(150) NIVEL_EDUCATIVO_ID int FECHA_INGRESO datetime2 CODIGO_MODULAR nvarchar(11)

Fuente: Elaboración propia.

Tabla Nº 11: Nomenclatura de Constraints (Restricciones)

NOMBRE	Primary key PK_<Nombre de la tabla> Foreign key FK_<Nombre de tabla origen>_<Nombre de tabla de referencia>_<identificador> Unique UQ_<Nombre de tabla> Default DF_<Nombre de columna> Check CH_<Nombre de tabla>_<Nombre de check>
DESCRIPCIÓN	Debe ser un nombre descriptivo y en MAYÚSCULA, sin caracteres especiales y no tiene un límite de caracteres establecido (recomendable no mayor de 30 caracteres) a excepción de los límites establecidos por el manejador de base de datos.
EJEMPLOS	PK_SERVIDOR_PUBLICO FK_SERVIDOR_PUBLICO_AFP DF_RESERVA_FECHA CK_PAGO_FECHA

Fuente: Elaboración propia.

Tabla Nº 12: Nomenclatura de Índices

NOMBRE	ix_<Nombre de tabla>_<Correlativo>
DESCRIPCIÓN	• Debe ser un nombre descriptivo, escrito en minúscula, sin caracteres especiales. • El correlativo debe tener tres dígitos rellenado con ceros a la izquierda. • Aplica si no se utiliza un modelador.
EJEMPLOS	ix_servidor_publico_001

Fuente: Elaboración propia.

Tabla Nº 13: Nomenclatura de Vistas

NOMBRE	vw_<Prefijo del sistema>_<Nombre Vista>_<Correlativo>
DESCRIPCIÓN	• Debe ser un nombre descriptivo, escrito en singular con minúscula y sin caracteres especiales. • En el caso que la vista sea una tabla única, se adopta el nombre de la tabla, en caso contrario se debe utilizar los criterios de las nomenclaturas de tablas.

EJEMPLOS	vw_sinad_001 vw_ubicacion_geografica

Fuente: Elaboración propia.

Tabla Nº 14: Nomenclatura de Procedimientos Almacenados

NOMBRE	USP_<Tipo lógico, >_<Nombre del procedimiento> Donde <Tipo lógico> es INS: Inserción UPD: Actualización DEL: Eliminación SEL: Consulta PRC: Procesamiento GEN: Genérico (tipo lógico diferente a INS, UPD, DEL; SEL y PRC) Por ejemplo: envío de email. **Nota:** Se utiliza el prefijo USP, que es un acrónimo de *User Stored Procedure*, para diferenciarlos de los procedimientos almacenados propios del motor de base de datos que inician con el prefijo SP.
DESCRIPCIÓN	• Debe ser un nombre descriptivo, escrito en singular con MAYÚSCULA y sin caracteres especiales. No tiene un límite de caracteres establecido, a excepción de los límites del manejador de bases de datos. • En el caso de tener un nombre compuesto, se deberá unir con el carácter *underscore*: "_". • Los nombres de los procedimientos almacenados no deben comenzar con el prefijo SP, para diferenciarlos de los procedimientos de sistema (*system procedures*). • Tampoco se debe usar el prefijo XP, que es un prefijo reservado para identificar los procedimientos almacenados extendidos.
EJEMPLOS	USP_INS_PERSONA USP_UPD_SERVIDOR_PUBLIC USP_DEL_PAIS USP_SEL_ASISTENCIA_DOCENTE

Fuente: Elaboración propia.

Tabla Nº 15: Nomenclatura de Funciones

NOMBRE	UFN_<Nombre de la función>
DESCRIPCIÓN	• Debe ser un nombre descriptivo de verbo en infinitivo con MAYÚSCULA y sin caracteres especiales. No tiene un límite de caracteres establecido, a excepción de los límites del manejador de bases de datos. • En el caso de tener un nombre compuesto, se deberá unir con el carácter *underscore*: "_". • Los nombres de las funciones no deben comenzar con el prefijo FN para diferenciarlos de las funciones de sistema (*system functions*).
EJEMPLOS	UFN_OBTENER_ESTADOS

Fuente: Elaboración propia.

Tabla Nº 16: Estructura del procedimiento almacenado

Todo procedimiento almacenado debe incluir lo siguiente:

- Nombre de procedimiento almacenado.
- Comentarios: Todos los procedimientos almacenados deberán tener los siguientes comentarios (Objetivo: historia de usuario, ticket de atención, Funcionalidad, descripción de variables entre otros).
- Declaración de parámetros *input* y/o *output.*
- Declaración de variables locales (en minúsculas).
- Sentencias *SQL*.

Consideraciones:

Palabras del lenguaje *SQL*, funciones y variables en MAYUSCULAS. Usar indentación y cada cláusula *SQL* debe ubicarse en una línea nueva.

Ejemplo:

```
-- ============================================
-- Procedimiento: ENVIO_EMAIL
--                      Modificación
-- Fecha                Autor           Descripción
-- 21/05/2021     Carlos La Rosa   Se adiciona parámetros
-- 10/06/2021     Juan Pérez       Se actualizar rutina de búsqueda
-- EXEC transaccion.PROGRAMAR_ENVIO_EMAIL
'avaron@minedu.gob.pe','','','High','Asunto','Mensaje <b>HTML</b>'
-- ============================================
ALTER PROCEDURE
[transaccion].[USP_PRC_PROGRAMAR_ENVIO_EMAIL]
        @para VARCHAR(MAX),
        @cc    VARCHAR(MAX),
        @cco  VARCHAR(MAX),
        @asunto       VARCHAR(MAX),
        @mensaje      VARCHAR(MAX)
AS
BEGIN
        SET NOCOUNT ON;
        BEGIN TRY
        EXEC [120.12.11.26].db_correo.dbo.USP_PRC_ENVIAR_CORREO
        'BUZON_ELECTRONICO',@para, @cc, @cco, 'High', @asunto,
@mensaje
        SELECT 1
        END TRY
        BEGIN CATCH
                SELECT 0
        END CATCH;
        SET NOCOUNT OFF;
    END
```

1.6.2 CONSIDERACIONES ESPECÍFICAS EN SQL

Para los *code snippets* de este apartado se ha tomado como referencia al motor *SQL Server*.

a. **Tipos de datos *SQL***
   - Optimizar el uso de los tipos de datos, de tal forma que se evite el exceso de espacio de almacenamiento.

- Para los tipos de datos caracteres se recomienda utilizar los tipos de dato *NCHAR* y *NVARCHAR*.
- Definir el tipo de dato correcto para una cadena de caracteres ya que para un *CHAR* (100) cuando es *NULL*, consumirá 100 bytes, lo que generará un desperdicio de espacio. Preferiblemente use *VARCHAR* (100). Las columnas (o variables) de longitud variable tienen una sobrecarga de procesamiento menor a las que se obtienen con columnas de longitud fija.
- En los tipos de datos *NVARCHAR* utilizar solo para longitudes mayores a 15 caracteres, normalmente aplicados a campos de descripción, nombres, etc. en donde no se sabe el tamaño de un texto determinado.
- Utilizar el tipo de datos datetime2 para caso de datos FECHA.
- Emplear el tipo de datos *TINYINT*, en el caso de tener columnas cuyo valor numérico no va a acceder de 255.
- No utilizar el tipo de datos IMAGEN. Utilizar campos *NVARCHAR* en donde se grabará la ruta donde se encuentra físicamente el archivo imagen.
- Evitar el uso de tipos de datos definidos por usuario (*User defined datatypes*). En caso ser necesario estos deberán ser debidamente sustentados y documentados.
- Se aceptan valores nulos en los casos que se requiera, pero se recomienda evitar que las columnas tengan valores *NULL*.
- Almacenar los objetos grandes en una tabla diferente, *VARCHAR*(*MAX*), *TEXT,* etc. Es preferible hacer dos lecturas para obtener los valores de objetos grandes.
- Evitar el uso de tipos de datos *TEXT* o *NTEXT* para almacenar datos textuales grandes, en su lugar utilice los caracteres máximos permitidos de *VARCHAR*.

b. **Consultas *SQL***

- Se recomienda evitar más de cinco *joins* en las consultas.
- En lugar de una gran consulta con muchos *joins*, dividir la consulta en varias consultas en tablas temporales de sesión y luego aplicar los *joins* sobre estas tablas más pequeñas.
- No utilizar "*SELECT **"
- Evitar el uso del *CASE* dentro de *SELECT*.
- Usar lo menos posible *ORDER BY*, esta es una de las operaciones más costosas en la ejecución de sentencias. Tampoco se debe utilizar números de columnas en las cláusulas *ORDER BY*.
- Evitar el uso de consultas que devuelvan más de 20KB en volumen, enviando al cliente sólo el bloque de datos que va a utilizar (visualizar) o paginar.
- Utilizar el nombre del esquema y nombre del objeto para el caso de las tablas.
- Tomar en cuenta los siguientes ejemplos y precisiones:

```
SELECT campo1, campo2 FROM dbo.mi_tabla
```

  - Al utilizar *LIKE* ingresar algunos caracteres, no debe estar vacío '%%', indicar un número de caracteres mínimos de modo obligatorio. Por ejemplo:

```
SELECT DISTINCT APELLIDOPATERNO, APELLIDOMATERNO,
NOMBRES
FROM dbo.persona
```

```
WHERE APELLIDOPATERNO LIKE 'ALVAR%'
```

- Utilizar una constante, en lugar de asterisco al realizar un *COUNT*. Por ejemplo:

```
SELECT @numero_productos = COUNT (1) FROM product
```

- No utilizar *IF* para condicionar varias sentencias, en lugar utilice *CASE* en una única sentencia. Por ejemplo:

```
USE db_basedatos
DECLARE @producto_id AS INT
SET @producto_id = 0
SELECT CAMPO1, CAMPO2
FROM dbo.producto
WHERE
PRODUCTO_ID = CASE @producto_id WHEN 0 THEN PRODUCTO_ID
ELSE @producto_id END
```

- No compare = *NULL*, en su lugar utilizar *IS NULL*. Por ejemplo:

```
SELECT CAMPO1, CAMPO2
FROM dbo.orden
WHERE REGION IS NULL
```

- Se recomienda no utilizar comparaciones con CAMPO='', en lugar utilice la combinación *ISNULL*() y *CASE*. Por ejemplo:

```
DECLARE @region NVARCHAR(15)
SET @Region = ''
SELECT CAMPO1, CAMPO2 FROM dbo.orden
WHERE ISNULL(REGION, '') = CASE WHEN @region = ''
THEN ISNULL(REGION, '') ELSE @region END
```

- Utilizar *EXISTS* en subconsultas y en lugar de traer un campo, utilice el escalar 1.

```
SELECT CAMPO1
FROM dbo.mi_tabla
WHERE EXISTS (SELECT 1 FROM dbo.segunda_tabla
WHERE mi_tabla.CAMPO2=segunda_tabla.CAMPO7)
IF EXISTS (SELECT 1 FROM sysobjects WHERE name = mi_tabla' AND
type = 'U')
```

- No usar *COUNT* en subconsultas. Por ejemplo:

```
SELECT CAMPO1, CAMPO2 FROM tabla WHERE 0 < (SELECT
COUNT(*) FROM tabla2 WHERE …)
```

- En lugar utilizar *EXISTS*

```
SELECT CAMPO1, CAMPO2
FROM table
WHERE EXISTS (
        SELECT CAMPO
        FROM table2
        WHERE ...)
```

- Tener cuidado en las condiciones que contengan tipos de datos de diferentes precisiones. En este caso se debe convertir al tipo de menor precisión. De presentarse esta situación, se puede generar errores de difícil detección. Por ejemplo:

```
SELECT CAMPO1, CAMPO2 FROM tabla_pequena, tabla_grande
WHERE tabla_pequena.FLOAT_COLUMNA =
tabla_grande.INT_COLUMNA
```

- Utilizar con cuidado sub consultas en lugar de funciones definidas por el usuario, estas últimas son más costosas en recursos.

- Concatenar, enumerar, formatear y convertir los tipos en la aplicación cliente, en lugar de la base de datos.

- Para mayor efecto del índice se debe incluir los campos índices como retorno de valor. Por ejemplo:

```
SELECT VENTA_FECHA, PERSONA_ID
FROM venta WHERE PRODUCTO_ID = 112
CREATE INDEX ix_venta_001 ON dbo.venta(PRODUCTO_ID)
INCLUDE(VENTA_FECHA, PERSONA_ID)
```

- Utilizar *NOT EXISTS* en lugar de *NOT IN*. Por ejemplo:

```
AND CAMPO3 NOT IN
(
SELECT CAMPO1
FROM tabla
WHERE CAMPO4 = @campo4
)
```

En su lugar utilizar:

```
AND CAMPO3 NOT EXISTS
(
SELECT CAMPO1
FROM dbo.tabla
WHERE CAMPO4 = @campo4
)
```

- Por cuestiones de legibilidad, se recomienda evitar el uso de <> como operador de comparación, en su lugar de usar el operador *IN*. Por ejemplo:

- ID <>2 (no recomendado)
- ID IN (1,3,4,5) (recomendado)

- En las consultas, usar “tablas derivadas” o *CTE (Common Table Expressions)* siempre que sea posible, ya que tienen un menor rendimiento. Por ejemplo:

```
SELECT MAX(PRODUCTO_PRECIO)
FROM producto
WHERE PRODUCTO_ID IN (
        SELECT TOP 2 PRODUCTO_ID
        FROM producto
        ORDER BY PRODUCTO_PRECIO DESC)
```

En su lugar utilice la consulta que se puede escribir utilizando una tabla derivada y generalmente se ejecuta dos veces más rápido que la consulta anterior.

```
SELECT MAX(PRODUCTO_PRECIO)
FROM (SELECT TOP 2 PRODUCTO_ID
        FROM producto
        ORDER BY PRODUCTO_PRECIO DESC)
```

- Evitar el uso innecesario de tablas temporales, en su lugar utilizar “tablas derivadas” o *CTE*, ya que tienen un mejor rendimiento.
- A menos que no se utilice mapeadores, no se debe escribir o actualizar directamente datos de tipo texto utilizándolas declaraciones *INSERT o UPDATE*. En su lugar tiene que usar declaraciones especiales como *READTEXT, WRITETEXT* y *UPDATETEXT*. Por lo tanto, si no tiene que almacenar más de 8 KB de texto, usar el tipo de datos *CHAR* (8000) o *VARCHAR* (8000) en su lugar.
- Utilizar una lista de columnas en sus declaraciones *INSERT* ya que esto ayuda a evitar problemas cuando la estructura de la tabla (como agregar o quitar una columna).
- Al ejecutar una instrucción *UPDATE o DELETE*, se debe usar la clave principal en la condición *WHERE* si es posible. Esto reduce las posibilidades de error.
- Evitar el *SQL* dinámico ya que tiende a ser más lento que el *SQL* estático. El motor debe generar el plan de ejecución en tiempo real.
- Incluir la instrucción *NOLOCK* en todas las sentencias *SELECT* a excepción de aquellas consultas cuyo resultado sirva para cálculos altamente consistentes, como por ejemplo cálculo de remuneraciones.

c. **Procedimientos almacenados y funciones *SQL***

- El uso de procedimientos almacenados solo está permitido en los siguientes casos:
  - Existen procedimientos almacenados complejos que ya están probados.
  - Su uso está comprobado genera mejoras sustanciales en el rendimiento del sistema.
- Los procedimientos almacenados deben ser los únicos que manipulen los datos (*SELECT, INSERT, UPDATE, DELETE*), debido a su mayor eficiencia.
- Utilizar "*SET NOCOUNT ON*" al inicio de los procedimientos almacenados.
- No utilizar cursores ni variables tipo tabla. Estos elementos utilizan memoria *RAM* del servidor en lugar de utilizar procesamiento de disco de las bases de datos *TEMPDB*.
- No utilizar funciones escalares dentro de *SELECT*, aplicada a cada columna.
- Usar *SP_EXECUTESQL*, en lugar de *EXEC* o *EXECUTE*, para sentencias *SQL* de tipo texto. Por ejemplo:

```
DECLARE @consulta NVARCHAR (100)
SET @consulta = N'SELECT CAMPO1, CAMPO2 FROM dbo.Persona
WHERE EDAD = @edad'
EXECUTE sp_executesql @consulta, N'@edad INT', @edad = 25
```

- Verificar siempre la variable global *@@ERROR* inmediatamente después de un *INSERT, UPDATE o DELETE*.
- Utilizar *TRY CATCH*. Por ejemplo:

```
BEGIN TRY
--Sentencias SQL
END TRY
BEGIN CATCH
--Error manejado
END CATCH
```

- Utilizar *RETURN* en caso haya algún dato que devolver.
- Usar subconsultas en lugar de cursores.
- Estandarizar el valor retornado en procedimientos almacenados. Si retorna un único registro, considerar la posibilidad de utilizar parámetros *OUTPUT*.

- Para estructurar las sentencias no se debe utilizar caracteres en blanco, en su lugar utilizar el carácter *TAB*.
- Las sentencias *SQL* deben escribirse de modo estructurado, reflejando secciones diferenciadas. Por ejemplo:

```
SELECT CAMPO1, CAMPO2 FROM tabla1, tabla2 WHERE tabla1.CODIGO = tabla2.CODIGO
```

En su lugar escribir:

```
SELECT
CAMPO1,
CAMPO2,
CAMPO3
FROM tabla1, tabla2
WHERE tabla1.CAMPO1 = Tabla2.CAMPO1
AND tabla1.CAMPO2 = Tabla2.CAMPO2
```

- En caso que no se utilice mapeadores, se recomienda devolver varios conjuntos de resultados de un procedimiento almacenado para evitar viajes del servidor de aplicaciones al servidor *SQL*.
- Considerar que la declaración *RETURN* está diseñada para devolver sólo el estado de la ejecución más no los datos.
- No utilizar *GOTO*.
- Siempre se debe sanitizar las variables filtrando caracteres especiales que permitan la inyección de código.
- La información confidencial de credenciales de acceso en especial contraseñas, no deben ser almacenadas directamente, éstas deben ser convertidas a sus valores hash y éstos ser almacenados.

d. **Optimización en *SQL***

- Durante la etapa de desarrollo se debe asegurar la optimización de sentencias *SQL* para evitar impactar negativamente el desempeño de la base de datos, se recomienda el uso de las herramientas de análisis de *tuning*, acompañadas de herramientas de pruebas de rendimiento y estrés
- Los campos con las condicionales de la cláusula *WHERE* deben ser en lo posible indexadas. Asimismo, no se debe realizar *JOINS* con columnas no indexadas.
- Prever que las columnas que se relacionan mediante *JOIN* deben tener el mismo *collation*.
- No utilizar vistas que usen una sola tabla.
- Usar tablas históricas y mover los datos por períodos diarios, semanales, mensuales, según se requiera.
- Utilizar siempre claves primarias y foráneas en las tablas, para garantizar integridad referencial.
- Evitar el uso de *DISTINCT*, asegúrese mediante condicionales que no existan duplicados o mediante índices *UNIQUE*.
- Seleccionar el campo de ordenamiento como candidato para índice.
- Utilizar *UNION ALL*, antes que *UNION*.
- Usar el plan de ejecución gráfica en el analizador de consultas o los comandos *SHOWPLAN_TEXT* o *SHOWPLAN_ALL* para analizar sus consultas.

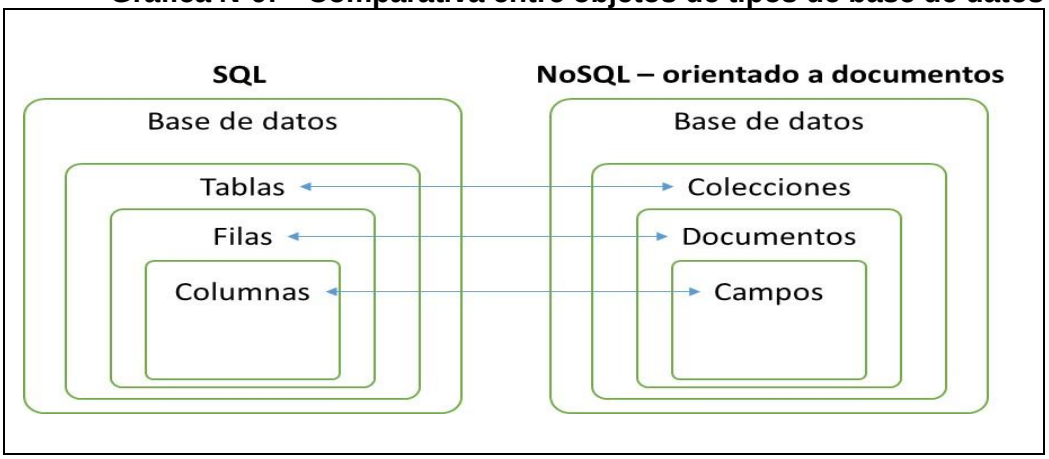
1.7 CONSIDERACIONES DE BASES DE DATOS *NOSQL*

1.7.1 CONSIDERACIONES GENERALES

El equipo de desarrollo debe tomar en cuenta las siguientes consideraciones relacionadas al uso de una base de datos *NoSQL*:

a. Las bases de datos orientadas a documentos tienen un esquema flexible de tal forma que las colecciones no necesariamente tienen una estructura uniforme de campos. Habiendo establecido el modelo conceptual y el tipo de base de datos *NoSQL*, es necesario tomar las decisiones que inciden en el rendimiento y en la legibilidad de los diseños. La siguiente gráfica muestra una comparación entre las bases de datos *SQL* y las *NoSQL* orientadas a documentos.

Gráfica Nº9: Comparativa entre objetos de tipos de base de datos



Fuente: Elaboración propia.

b. Debe aplicar las siguientes nomenclaturas:

Tabla Nº 17: Nomenclatura de Bases de Datos *NoSQL*

DESCRIPCIÓN	Se debe usar camelCase. El nombre no debe contener espacios en blanco. Debe incluirse el sufijo DB. Para base de datos de servicios (cuando se usa el patrón “database per service”).: Se nombran con el prefijo conteniendo al inicio el nombre de la aplicación seguido del nombre del recurso o entidad principal o dominio.
EJEMPLOS	Base de datos de aplicaciones: - siagieDB - sigepDB - sigesDB Base de datos de servicios: - spepCommentsDB (base de datos para el servicio de comentarios en el sistema SPEP). - ayniCalculationsDB (base de datos para el servicio que ejecuta los cálculos de descuentos en el sistema AYNI).

Fuente: Elaboración propia

Tabla Nº 18: Nomenclatura de Colecciones

DESCRIPCIÓN	Se nombran en inglés, de preferencia; o en castellano (sin mezclar los idiomas) usando l*owerCamelCase*[12]. El nombre no debe contener espacios en blanco. Si se elige el idioma inglés, los nombres de colecciones deben nombrarse en plural. En castellano, podrán ir en singular.
EJEMPLOS	Colecciones en inglés: - comments - calculations - periods - notifications - students - asssets Colecciones en castellado: - informeControl - materialEducativo

Fuente: Elaboración propia

Tabla Nº 19: Nomenclatura de Campos

DESCRIPCIÓN	Se debe usar camelCase. No se debe usar el carácter underscore (“_”) como carácter de inicio en el nombre del campo. Se debe usar inglés o castellano de forma uniforme, según el idioma elegido como nombre de las colecciones.
EJEMPLOS	Para la colección “students” - fullName - birthDate - father, que a su vez contiene un objeto con los campos: fullnName, birthDate, documentType, documentNumber. Para la colección “informeControl” - fechaEmision - remitente, que a su vez contiene el objeto con los campos: nombre, tipoDocumento, numeroDocumento.

Fuente: Elaboración propia

1.7.2 MODELO DE DATOS EMBEBIDOS *NOSQL*

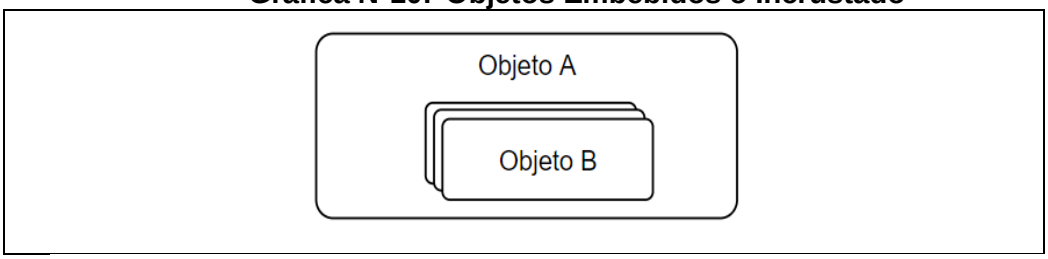
a. Este modelo de datos se debe usar cuando:
   - Se tiene relaciones de tipo “contiene a” entre entidades, la que aplica a relaciones entre documentos de uno a uno.
   - Se tiene relaciones de tipo “uno a muchos” donde los documentos hijo (“muchos”) siempre pueden ser accedidos desde el contexto del documento padre.

[12] Camel case, es un estilo de escritura que se aplica a frases o palabras compuestas. También se le conoce como “capitalización medial” en la que, por ejemplo “Institución Educativa” pasa a ser “institucionEducativa” o “InstitucionEducativa”. LowerCamelCase implica que la primera letra va en minúsculas, por ejemplo: “institucionEducativa”.

b. Este modelo es recomendable cuando:
   - Los subdocumentos (o documentos embebidos) son pequeños (menor a 16 MB) y pocos.
   - Los datos no cambian con frecuencia.
   - Cuando es aceptable tener consistencia eventual.
   - Se necesita alta capacidad de lectura.

c. No es recomendable cuando se presenten situaciones donde los documentos crecen después de su creación. El crecimiento de los documentos puede impactar en el rendimiento de escritura y llevar a fragmentar los datos. Gráficamente se puede reconocer con este diagrama:

Gráfica Nº10: Objetos Embebidos o Incrustado



Fuente: Elaboración propia.

Por ejemplo, en este *JSON*, los estudios de una persona están embebidos dentro de la etiqueta “estudios”:

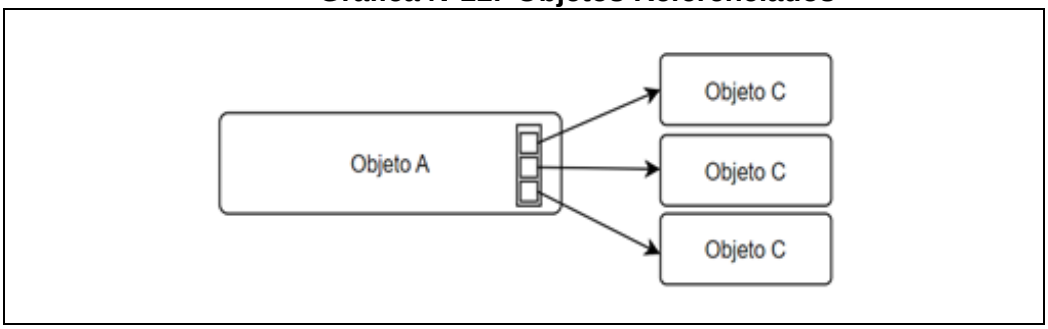
```
{
  "_id": 1,
  "dni": "18059307",
  "nombre": "Claudio Obando",
  "direccion": {
    "calle": "Doña Nelly",
    "numero": "120",
    "distrito": "Surco",
    "provincia": "Lima",
    "region": "Lima"
  },
  "estudios": [
    {
      "centro": "Universidad de Lima",
      "carrera": "Administración",
      "egreso": 2010
    },
    {
      "centro": "Universidad Católica",
      "carrera": "Psicología",
      "egreso": 2016
    }
  ]
}
```

1.7.3 MODELO DE DATOS CON REFERENCIAS NO SQL

a. En general se debe usar este modelo para:
   - Cuando el modelo embebido significa duplicación de datos perjudicando las operaciones de lectura.
   - Representar relaciones complejas “muchos a muchos”.
   - Para modelar data sets jerárquicos grandes.
   - Las referencias dan mayor flexibilidad que los embebidos, aunque implican varias *queries* desde la capa cliente, y por tanto más idas y vueltas al servidor.

Gráficamente se puede reconocer con este diagrama:

Gráfica Nº11: Objetos Referenciados



Fuente: Elaboración propia.

Por ejemplo, se tienen los siguientes documentos con modelo embebido:

ESTUDIANTES Y SUS MATRÍCULAS EN COLEGIO	

{
"_id": ObjectId("507f1f77bcf86cd799439123"),
"estudiante": "Mariela Salas",
"edad": 10,
"colegio": {
"nombre": "Miguel Grau",
"direccion": "Av. América 352",
"provincia": "Trujillo"
}
}
	{
	"_id":
	ObjectId("507f1f77bcf86cd799439456"),
	"estudiante": "Eduardo Quiñones",
	"edad": 12,
	"colegio": {
	"nombre": "Miguel Grau",
	"direccion": "Av. América 352",
	"provincia": "Trujillo"
	}
	}

Es posible transformarlo al siguiente modelo referencial.

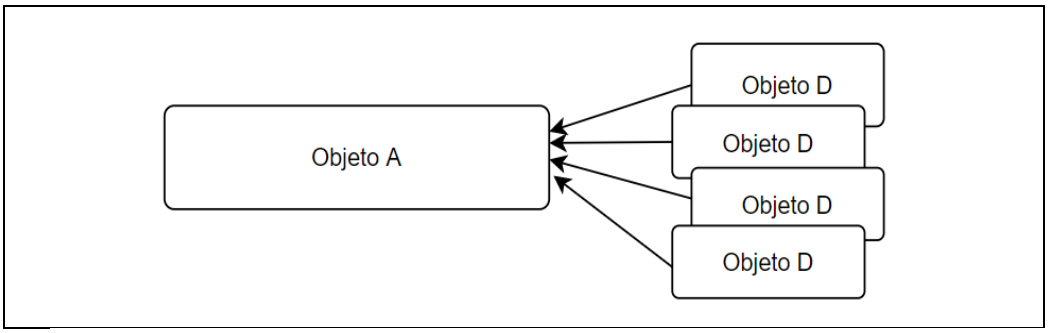
MATRÍCULA EN COLEGIOS	ESTUDIANTES

{
"_id": ObjectId("507f1f77bcf86cd799439001"),
"nombre": "Miguel Grau",
"direccion": "Av. América 352",
"provincia": "Trujillo",
"estudiantes": [
ObjectId("507f1f77bcf86cd799439011"),
ObjectId("507f1f77bcf86cd799439012")
]
}
	{
	"_id":
	ObjectId("507f1f77bcf86cd799439011"),
	"estudiante": "Mariela Salas",
	"edad": 10
	}
	{|
	"_id":
	ObjectId("507f1f77bcf86cd799439012"),
	"estudiante": "Eduardo Quiñones",
	"edad": 12
	}

b. Este modelo es recomendable cuando:
   - Los subdocumentos (o documentos embebidos) son muchos.
   - Los datos cambian con frecuencia.
   - Cuando la consistencia inmediata es necesaria.
   - Se necesita alta capacidad de escritura.

   Una variante de este modelo se refleja cuando los objetos hijos referencian al objeto padre. Gráficamente es como sigue:

Gráfica Nº12: Variante de Referencia



Fuente: Elaboración propia

1.8 CLASIFICACIÓN DE BASE DE DATOS *NOSQL*

a. Bases de datos orientadas a documentos
   Se consideran aquellas bases de datos que gestionan datos semiestructurados o sin un esquema único. Los datos son almacenados en algún formato estándar como puede ser *XML, JSON o BSON*. Son las bases de datos *NoSQL* más versátiles. Se pueden utilizar en diversos desarrollos, incluyendo muchos que tradicionalmente funcionarían sobre bases de datos relacionales. En esta categoría se encuentran, por ejemplo:
   - MongoDB.
   - CouchDB.

b. Bases de datos orientadas a columnas.
   Se consideran aquellas bases de datos que permiten realizar consultas y agregaciones sobre grandes cantidades de datos. Funcionan de forma parecida a las bases de datos relacionales, pero almacenando columnas de datos en lugar de registros. En esta categoría se encuentran, por ejemplo:
   - Cassandra.
   - HBase.

c. Bases de datos de clave valor.
   Se consideran aquellas bases de datos que guardan tuplas que contienen una clave y su valor de tal forma que cuando se requiere recuperar un dato, simplemente se busca por su clave y se recupera el valor. En esta categoría se encuentran, por ejemplo:
   - DynamoDB.
   - Redis

d. Bases de datos en grafo.
   Se consideran aquellas bases de datos que se basan en la teoría de grafos, donde se utilizan nodos y aristas para representar los datos almacenados. Son muy útiles

para guardar información en modelos con muchas relaciones, como redes y conexiones sociales. En esta categoría se encuentran, por ejemplo:

- Infinite Graph.
- Neo4j.

ANEXO Nº 05
ESPECIFICACIONES Y CONSIDERACIONES TÉCNICAS PARA EL USO DE INTERFACES DE PROGRAMACION DE APLICACIONES (v1.0)

1.1 DEFINICIONES

- ***API Management*:** Es el proceso de crear y publicar un *API web*, orientado al uso de políticas, controlando el acceso, coleccionando y analizando las estadísticas de uso y reportando el performance. El administrador de las *API* es el rol encargado en el despliegue, monitoreo, optimización, mantenimiento y baja.
- ***Back-end*:** Es la capa de cada sistema informático que se encarga del acceso a los datos. El *back-end* no es directamente accesible por los usuarios y además contiene la lógica de la aplicación.
- **Balanceo de carga:** Consiste en la práctica de distribuir las peticiones del cliente a través de múltiples instancias de la aplicación, siempre que éstas se encuentren operativas y evitando de esta forma que una simple instancia se vea sobrecargada. El balanceo de carga también permite escalar aplicaciones y actualizarlas sin necesidad de tener tiempos de inactividad total de la aplicación. Esto se logra habilitando un despliegue continuado en una pequeña cantidad de instancias antes de implementarlo totalmente.
- ***Case sensitive*:** Es una expresión usada en informática que se aplica a los textos, donde su procesamiento es sensible a mayúsculas o minúsculas.
- ***Claim*:** En el marco de los sistemas de seguridad, es una pieza de información sobre un sujeto u organización, por ejemplo, puede ser sobre un nombre, grupo, rol, privilegio, asociación o nivel de permisos. Se representa generalmente como un conjunto clave/valor.
- ***Endpoint*:** Es un extremo de un canal de comunicación. Cuando una *API* interactúa con otras *API*, los puntos de contacto de esta comunicación se consideran "*endpoint*". Un *endpoint* puede incluir una *URL* de un servidor o servicio. Cada punto final es la ubicación desde la cual las *API* pueden acceder a los recursos que necesitan para llevar a cabo su función.
- ***Front-end*:** Es parte visible del desarrollo web o del lado del usuario final, que interactúa con los usuarios, es conocida como el lado del cliente. Son aquellas tecnologías de diseño y desarrollo web que se ejecutan en el navegador y que se encargan de la interactividad con los usuarios como por ejemplo botones, menús, páginas, enlaces, gráficos y otros componentes de una página.
- **Idempotente:** Aplicado a *REST* (realmente es a los métodos *HTTP*) significa que la ejecución repetida de una petición con los mismos parámetros sobre un mismo recurso tendrá el mismo efecto en el estado de nuestro recurso en el sistema si se ejecuta 1 o N veces.
- ***JWT*:** Es un estándar RFC 7519/7523 para transmitir información con la identidad y "*claims*" de un usuario de forma segura entre un cliente/servidor. Dicha información puede ser verificada y confiable porque está firmada digitalmente.
- **Métodos *HTTP*:** Son palabras o comandos que permiten comunicar al servidor lo que se quiere realizar con un recurso bajo una *URI*. Los métodos más importantes de *HTTP* (especialmente para hacer aplicaciones *REST*) son *POST, GET, PUT, DELETE, OPTIONS* y *HEAD*.
- **Modelo de madurez de Richardson:** Modelo propuesto por Leonard Richarson que permite entender el concepto *REST* y por tanto conseguir llevar a cabo mejores implementaciones del mismo.

- **Nivel 0 - Protocolo *HTTP:*** Este es el nivel inicial en el que está referido sólo el uso del protocolo HTTP para realizar interacciones remotas, pero sin usar otros mecanismos existentes para la web. Básicamente se usa *HTTP* como un mecanismo de *tunneling* para tu propio mecanismo de interacción, normalmente basado en *RMI*.
- **Nivel 1- Recursos**: El primer paso para conseguir el nivel de madurez es introducir los recursos. Este nivel el objetivo es identificar recursos a través de una *URI* sin especificar la acción a realizar sobre el mismo.
- **Nivel 2- Uso de verbos**: Hasta ahora en los niveles anteriores se ha podido estar usando para las peticiones el verbo *HTTP POST*. Este nivel nos indica que la *API* debería utilizar los verbos *HTTP*, mencionados anteriormente. Utilizando correctamente los verbos y las respuestas.
- **Nivel 3- Uso de controles de hypermedia**: Básicamente es utilizar *HATEOAS* (Hipertexto como el mecanismo del estado de la aplicación). Lo que indica *HATEOAS* es que al realizar un *request*, el mismo nos retorne información de como trabajar o manipular el recurso.

- **Parámetros de cabecera:** Los parámetros son opciones que se pasan a través del *end-point* (como por ejemplo especificando el formato o la cantidad de recursos a ser retornados) para influenciar en la respuesta.
- **Patrón:** En el marco del desarrollo de sistemas, patrón es una solución general reusable que puede ser aplicada a problemas que ocurren comúnmente en el desarrollo de *software*. Es la descripción o plantilla de cómo resolver un problema que puede ser usado en diferentes situaciones.
- **Patrón arquitectónico:** Solución general y reutilizable a un problema común en la arquitectura de software dentro de un contexto dado. Los patrones arquitectónicos son similares al patrón de diseño de *software*, pero tienen un alcance más amplio.
- **Patrón de diseño:** Es una técnica para resolver un problema común en el desarrollo de *software* y otros ámbitos referentes al diseño de interacción o interfaces. Para que una solución sea considerada un patrón debe poseer ciertas características, una de ellas implica haber comprobado su efectividad resolviendo problemas similares en ocasiones previas. Asimismo, debe ser una solución reutilizable.
- **Patrón *Throttling:*** Es un patrón arquitectónico que permite controlar la cantidad de recursos (servicios) disponibles hasta cierto nivel y después limitar o estrangular servicios no priorizados para prevenir el incumplimiento de los *SLA* o evitar el colapso del sistema.
- ***REST*:** *Representational State Transfer* (en español la Transferencia Representacional de Estado). Es una técnica o estilo de arquitectura de *software* usada para construir *API*s que permitan comunicar servidores con sus máquinas clientes usando el protocolo *HTTP* mediante *URIs* definidas. *REST* es más simple y convencional que otras alternativas que se han usado como *SOAP y XML-RPC*.
- ***URI*:** *Uniform Resource Identifier* (en español Identificador Uniforme de Recursos). La *URI* no se debe confundir con el *URL*, ya que las *URI* incluyen en su estructura a una *URL*. La diferencia fundamental es que mientras las URI identifican, las *URL* localizan. La estructura de un *URI* es como sigue:

  {protocolo}://{dominio o hostname}[:puerto (opcional)]/{ruta del recurso}?{consulta de filtrado}

- ***URL*:** *Uniform Resource Locator* (en español Localizador Uniforme de Recursos). Es la dirección completa de una página web, mientras que la dirección del servidor se conoce como *DNS*. La *URL* es la dirección electrónica para poder acceder a un recurso en un servidor remoto (en un puerto específico). El tipo más común de *URL* es el de las páginas *web*, con la dirección *http*://, pero existen otras direcciones *URL* como *ftp*://, que proporciona la ubicación de red de un recurso *FTP* para poder transferir archivos.
- ***UUID*:** El identificador único universal es un número de 16 bytes (128 bits). Por tanto, el número de posibles *UUID* es de 1632, o unos 3,4 × 1038. En su forma canónica un *UUID* se expresa mediante 32 dígitos hexadecimales divididos en cinco grupos separados por guiones de la forma 8-4-4-4-12 lo que da un total de 36 caracteres (32 dígitos y 4 guiones). Por ejemplo: 550e8400-e29b-41d4-a716-446655440000.

1.2 CLASIFICACIÓN DE LAS *API*

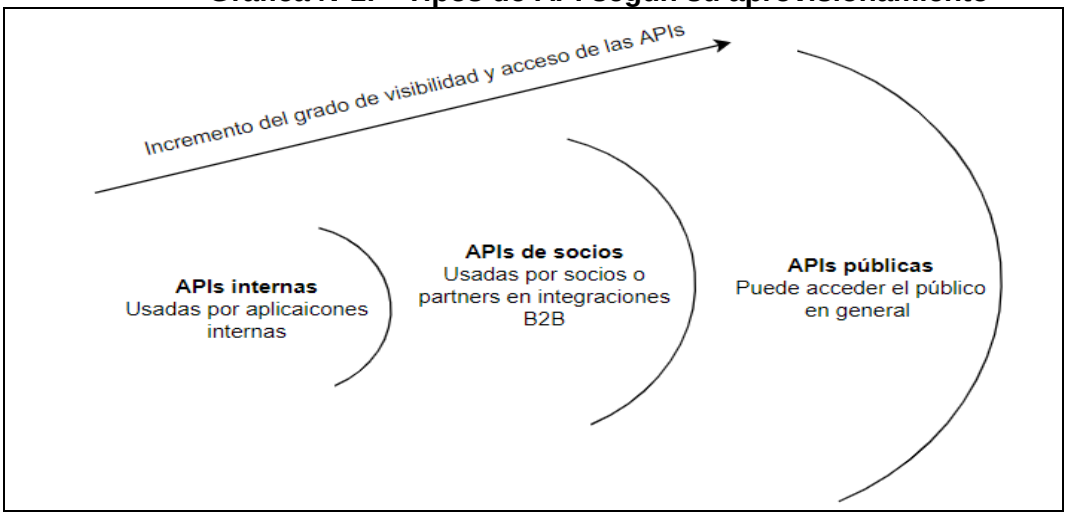
1.2.1 SEGÚN SU APROVISIONAMIENTO: Esta clasificación viene desde la perspectiva de quién usa las *API*:

i. ***API* pública.** Son las que puede ser usadas por una comunidad o público en general. Por lo general, son gratuitas, pero con frecuencia están sujetas a un cobro por su uso. Por ejemplo, la *API* que permite la consulta de boletas.

ii. ***API* de socios (o de *partners*).** Son las que puede ser usadas por un grupo selecto de procesos previamente registrados en el marco de colaboraciones *B2B*. Por ejemplo, la *API* que se consume para la obtención de los datos personales desde RENIEC.

iii. ***API interna* (o privada).** Son las que solo pueden ser usadas desde el interior de una organización, para la atención de procesos automáticos. Por ejemplo, la *API* del servicio que obtiene los estudiantes matriculados en una institución educativa que es utilizado para el envío de invitaciones para cursos gratuitos.

Gráfica Nº1: Tipos de API según su aprovisionamiento



Fuente: Elaboración propia.

1.2.2 SEGÚN SU ESTILO DE DISEÑO: Según este tipo de clasificación, el tipo más utilizado son las *API* de tipo *REST*, en corto: *API REST*. Sin embargo, debemos indicar la existencia de otros tipos de *API: GraphQL* que permite recuperar información a demanda, en contraposición de las capacidades de *REST* que adolece de esta capacidad.

1.2.3 SEGÚN EL FORMATO DE DATOS: Según este tipo de clasificación, mencionaremos a las *API* de tipo *JSON*, que son los tipos principalmente utilizados en el presente documento, sin embargo, se debe reconocer la existencia de una *API* en formato *XML* y en formato *YAML*[13] es el tipo más utilizado son las *API REST*.

1.2 VENTAJAS DE USAR UNA *API REST*

API REST permitirá aprovechar las siguientes ventajas en el desarrollo de aplicaciones distribuidas en el MINEDU:

a. **Agnóstico a la tecnología:** Es posible implementar una *API REST* utilizando diferentes herramientas, *frameworks*, lenguajes de programación y bases de datos.
b. **Mejora en la performance:** *REST* no usa sesiones, por lo que se dispone de mayor cantidad de memoria *RAM*. Asimismo, usa archivos *JSON*, en lugar de *HTML*, por lo que las respuestas al cliente tienen menor volumen.
c. **Escalabilidad:** Es un atributo de calidad muy buscado en la toma de decisiones arquitectónicas para productos de *software*. Una de las formas más recurrentes para aumentar los niveles de escalabilidad de un producto de *software* es a través del uso de servicios o microservicios que se comunican a través de sus *API*.
d. **Separación entre el *back-end* y *front-end*:** Entre las ventajas se encuentra la posibilidad de separar y especializar los equipos de desarrollo en diferentes grupos que utilicen como mecanismo de interacción a las *API*. El equipo de *back-end* construye, orquesta y despliega las *API* y el equipo de *front-end* las consume.

1.3 RESTRICCIONES DE REST (COMO ESTILO DE ARQUITECTURA)

API REST es un estilo de arquitectura constituido por un conjunto de restricciones con el que se pueden crear aplicaciones distribuidas sobre *HTTP*. Las siguientes son las restricciones:

a. **Interface uniforme:** También conocido como “contrato uniforme”, indica que un cliente antes de interactuar con un servicio *REST* debe acordar la identificación y la manipulación. Mediante la identificación, se especifica que debe ser posible conocer cada recurso que el servicio tiene para ofrecer. Mediante la manipulación se establece que existen un conjunto estándar de operaciones que se pueden aplicar sobre cualquier recurso con resultados predecibles.
b. **Cliente servidor:** En el marco de un desarrollo distribuido de aplicaciones, la aplicación cliente y la aplicación servidor deben evolucionar por separado y de forma independiente entre sí.
c. **Sin estado:** Todas las interacciones entre el cliente y el servidor no tiene estado. Si una aplicación requiere que el servidor recuerde su estado, como por ejemplo cuando un usuario inicia sesión una vez y realiza otras operaciones autorizadas

[13] YAML es un acrónimo recursivo que significa ***YAML Ain't Markup Language*** (en castellano, 'YAML no es un lenguaje de marcado').

después de eso, entonces; cada solicitud de la aplicación cliente debe contener toda la información necesaria para que el servidor atienda la solicitud.

d. **Caché:** El almacenamiento en caché se aplicará a los recursos cuando se requiera mejorar el rendimiento del sistema. Para esto, los recursos deben declararse almacenables en caché. El almacenamiento en caché se puede implementar en el servidor o en el lado del cliente.
e. **Sistema en capas:** Un sistema en capas exige una separación de responsabilidades, lo cual significa a su vez que cada capa interactúe con su capa adyacente sin pasar por alto alguna capa.

1.4 CONSIDERACIONES PARA LA ELABORACIÓN DE UNA *API REST*

1.4.1 NOMBRES EN LA *API REST*

a. Cada *API REST* debe ser implementada para que pueda ser utilizada con facilidad por otros analistas y desarrolladores, por lo que debe ser una *API* simple de tal forma que un desarrollador deba recurrir a la documentación lo menos posible.
b. A menos que sea requerido de otra forma, todas las *API* deben estar implementadas utilizando en su nombre el idioma español, evitando los caracteres especiales tales como vocales acentuadas, la letra “ñ”.
c. Se debe evitar el uso de abreviaturas en los nombres de los recursos y los parámetros. Por ejemplo, no use en la *API*: “f*acts*” (por indicar facturas), “*recingr*” (por indicar recibos de ingreso), entre otros.
d. Para nombrar los recursos y sus atributos se debe usar el estilo *spinal-case*. Una breve descripción de los diferentes estilos se encuentra en el numeral 7.4.7. Asimismo, acorde a lo indicado en el *RFC* 3986 *Uniform Resource Identifier* (*URI): Generic Syntax,* las *URL* de las *API* son *case sensitive*, excepto el esquema y el *hos*t.
e. Como nomenclatura en el cuerpo del *request* y en el *response*: Utilizar *lowerCamelCase* para las etiquetas del *JSON* tanto del *body* como del response respectivamente, por ejemplo:

```
{
"conceptoId": "34",
"conceptoDescripcion": "Pago de matrícula"
}
```

f. No utilizar extensiones de archivos, en su lugar utilizar la cabecera *Content-Type* y *Accept.* Por ejemplo, la *URI* /boleta-notas/234.pdf no es correcta.
g. Se debe usar la siguiente nomenclatura de la *URI* base.

https://<ambiente>-<producto>.<dominio>:<puerto>/<version>/<sub-dominio>/<recursos>

Tabla Nº 1: Nomenclatura de la *URI* Base

ETIQUETAS	DESCRIPCIÓN
<ambiente>	Indica los ambientes en los que están desplegados los servicios: Desarrollo: “dev-api” Control de calidad (certificación): “qa-api” Producción: “api”
<producto>	Indica el producto o sistema, generalmente siglas o acrónimos; al que se encuentra asociado el servicio, por

ETIQUETAS	DESCRIPCIÓN
	ejemplo: “siagie”, “sisda”, “sinad”, “sigas”, “planin”, entre otros.
	Le sigue una identificación del servicio que contiene al conjunto de *API*, considerando incluir, en lo posible como máximo dos palabras separadas por un guion, por ejemplo: “configuracion”, “matricula-extemporanea”, “boleta-electronica”.
<dominio>	minedu.gob.pe Nota**:** Antes del dominio no debe incluirse ningún punto.
<puerto>	Considerar usar el mismo puerto para el mismo producto en los diferentes ambientes.
<version>	v1, v2, v3, v4
<sub-dominio>	Es el nombre del servicio implementado, por ejemplo: “identidad”, “evaluacion”, “capacitacion”, “gestion”, “contratacion”
<recursos>	Son los recursos, por ejemplo: “estudiantes”, “docentes”, “equipos”, “instalaciones”, “materiales”.

Fuente: Elaboración propia

Por ejemplo:

- dev-api-sisda-boletas.minedu.gob.pe/v1/pagos/planillas
- dev-api-sifods-capaciontaciones.minedu.gob.pe/v1/capacitacion/docentes
- qa-api-sifods-capacitaciones.minedu.gob.pe/v2/gestion/materiales
- api-sisda-calculo-planilla.minedu.gob.pe/v1/pagos/planillas

1.4.2 USO DE SUSTANTIVOS EN PLURAL PARA NOMBRAR RECURSOS

a. El criterio para elegir el sustantivo correcto para la *API*, es el lenguaje ubicuo aplicado en el proyecto de desarrollo de software con una orientación al dominio.
b. Se debe utilizar el sustantivo en plural que describe cada recurso:

Tabla Nº 2: Ejemplos de uso de sustantivos en plural RECURSO

Ejemplos	VERBOS HTTP			
	POST	GET	PUT	DELETE
/cursos	Crear un curso	Mostrar todos los cursos	Actualizar los datos de cursos	Borra todos los cursos
/cursos/a182h1a1	No aplica	Mostrar el curso con el *uuid* a182h1a1.	Actualizar el curso con el *uuid* a182h1a1.	Borrar el curso con el *uuid* a182h1a1.
/docentes	Dar de alta a un docente	Mostrar todos los docentes.	Actualizar los datos de docentes.	No aplica.

/docentes/134	No aplica	Mostrar el curso con el id 134.	Actualizar el docente con el id 134.	Borrar el docente con el id 134.

Fuente: Elaboración propia[14]

1.4.3 USO DE MÉTODOS *HTTP*

a. Se debe tomar en cuenta que los métodos *HTTP* proporcionan la contraparte de la acción del recurso basado en sustantivos y que *HTTP* define un conjunto de métodos de petición para indicar acciones a realizar para determinados recursos.
b. El uso de métodos *HTTP* requiere manejar dos conceptos importantes: Métodos seguros y Métodos idempotentes.
   - Un método *HTTP* es considerado seguro si no altera el estado de la aplicación *back-end*. En otras palabras, un método es seguro si conduce a una operación de 'sólo lectura' cuyo resultado es repetible, por ejemplo: *OPTIONS, GET o HEAD*. Todos los métodos seguros son también a su vez idempotentes.
   - Un método *HTTP* es idempotente si al realizar una operación muchas veces da el mismo resultado cual si se hubiese realizado la operación una sola vez, en otras palabras; si una solicitud *HTTP* idéntica puede realizarse una o demasiadas veces consecutivamente obteniendo el mismo resultado dejando al servidor en el mismo estado.

Tabla Nº 3: Métodos *HTTP* utilizados con más frecuencia

MÉTODO	OPERACIÓN	ES IDEMPOTENTE	ES SEGURO
POST	Creación	No	No
GET	Lectura	Si	Si
PUT	Actualización/Reemplazar	Si	No
PATCH	Actualización/Modificar	No	No
DELETE	Eliminación	Si	No
HEAD	*GET* sin cuerpo de respuesta	Si	Si
OPTIONS	Describe operaciones de comunicación	Si	Si

Fuente: Elaboración propia

c. Los códigos de estado de respuesta *HTTP* a utilizar en el MINEDU están definidos en el *RFC* 2616 - *Hypertext Transfer Protocol* (*HTTP*/1.1) y no deberán personalizarse:
   - 1xx (informativo): la solicitud se recibe y continúa procesándose.
   - 2xx (correcto): la solicitud se recibió, se comprendió y se aceptó correctamente.
   - 3xx (redirección): se deben realizar más acciones para completar la solicitud.
   - 4xx (Error del cliente): la solicitud contiene la sintaxis incorrecta o no se puede cumplir.
   - 5xx (Error del servidor): el servidor no cumple con una solicitud aparentemente válida.

[14] El uso correcto de recursos en una API REST permite establecer la ruta de adopción del Modelo de Madurez de Richardson.

1.4.4 VERBOS PROPIOS DE *HTTP/HTTPS* EN LA *URI*

a. Se debe evitar el uso de verbos en la *URI* tales como: "*get", "save*", "grabar", "obtiene", "lista", etc, por ejemplo:

GET /getUsers	DELETE /borrarCurso	PUT /setNewUsuario
POST /aplicarCambio	GET /conseguir	PUT /updateEstudiante

b. Se puede utilizar verbos como "_sort", "_limit", "_start", en el ámbito del uso de filtros u ordenadores. Por ejemplo:

GET /estudiantes?_**sort**=email:ASC
GET /colegios?_**start**=10&_**limit**=10

1.4.5 APLICACIÓN DE VERBOS QUE DENOTAN ACCIONES

a. Se puede utilizar verbos que denotan acciones, siempre que no denoten operaciones *CRUD*, por ejemplo: activar, desactivar, matricular, confirmar, firmar, priorizar, cerrar, entre otros. Estos verbos, deben ser siempre utilizados al final de la *URI* y aplica para un único recurso o para la colección.

Tabla Nº 4: Ejemplo de verbos

EJEMPLO	DESCRIPCIÓN
/estudiantes/as3jas123/matricular	Matricula al estudiante con el *uuid* as3jas123
/colegios/812as13/cerrar	Cierra el colegio con el *uuid* 812as13
/colegios/2020/iniciar	Inicia todo el periodo anual de los colegios.

1.4.6 RELACIONES ENTRE RECURSOS

a. Se puede realizar operaciones sobre recursos relacionados, por ejemplo, cuando requerimos construir una *API* que relacione dos recursos como por ejemplo cursos y estudiantes. Por ejemplo:

…/cursos/{id}/estudiantes
Obtiene todos los estudiantes que se han matriculado en el curso con identificación ":id".

b. El *slash* ("/") debe usarse para indicar una relación jerárquica, al final de una *URI* no aporta nada y no se debe incluir.

/cuadernodecontrol.minedu.gob.pe/cuadernos /cuadernodecontrol.minedu.gob.pe/cuadernos/{id}/dias/{id}

c. En la *URL*, sólo se debe relacionar dos recursos como máximo, para mantener la característica de simplicidad en la *API*.
d. Para la implementación de relaciones con un mayor número de recursos, se debe implementar un nuevo *URI* o utilizar el pase de parámetros por el operador "?".

1.4.7 USO DE PARÁMETROS DE CABECERA (*HEADERS*)

a. *API REST* maneja cuatro tipos de parámetros: De cabecera, De ruta, Cadena de consulta (*query strin*g) y Cuerpo de pedido (*request body*).
b. Los encabezados *HTTP* son una parte importante de la petición y de la respuesta de la *API* ya que llevan la información siguiente:
   - Sobre el cuerpo de la petición y la respuesta
   - La autorización de la petición.
   - El manejo en *caché* de la respuesta.

- El manejo de *cookies* en la respuesta.

c. Los encabezados *HTTP* contienen otra información como los tipos de conexión *HTTP*, servidores *proxy*, etc. La mayoría de estos encabezados son para la administración de conexiones entre cliente, servidor y servidores *proxy* y no requieren validación explícita mediante pruebas.
d. Se debe evitar el uso de caracteres especiales en los valores de los parámetros de cabecera.

1.4.8 MANEJO DE VERSIONES DE UNA *API* FRENTE A LA IMPLEMENTACIÓN DE NUEVAS VERSIONES

a. Se debe realizar un trabajo coordinado entre los integrantes del equipo de desarrollo del *back-end* y de los de *front-end*
b. El manejo de versiones de una *API* se debe implementar sólo a través de tres grupos: Por *URL* o Por *Header* o por *Media Types*.
c. Se debe cumplir la siguiente secuencia de actualización[15]:
   - Se cuenta con un servicio en el back-end consumido por cuatro tipos de cliente a través de una *API* con versión v1.
   - Se requiere una actualización de la lógica del servicio que a su vez requiere el cambio de la *API*. La *API* cambia a una versión v2.
   - Progresivamente, se va migrando los tipos de clientes hacia el nuevo servicio.
   - Al final, todos los clientes consumen el servicio a través de la *API* v2.
d. El esquema de versionado que debe ser adoptado para las *API* es el versionado por *URL*. Aunque existen diferentes formas de utilizar este esquema de versionado, el siguiente es una buena práctica a aplicar:

.../mapas/version/2	No utilizar
.../mapas/version/2/escuelas/version/3	No utilizar
.../mapas/v2/escuelas	**Utilizar**

e. Para obtener el número de versión que corresponde a la API se debe usar versionamiento semántico[16].

1.4.9 MANEJO DEL PARAMETROS DE REQUEST

a. Los atributos definidos como obligatorios se deben validar siempre en el servicio.
b. Los atributos definidos como opcionales, se deben validan solo si tienen contenido, caso contrario se deben ignorar.
c. Los atributos enviados que no son parte del esquema o de la definición, deben ser completamente ignorados.

1.4.10 ESTRUCTURA DE LAS RESPUESTAS *JSON*

a. Se debe uniformizar la estructura de los *JSON* de respuesta (*response*), a fin de poder disminuir el esfuerzo en el entendimiento de los desarrolladores de *back-end* y *front-end.*
b. Para la distinción de las *respuestas exitosas* con las *respuestas de error*, se debe utilizar los códigos de estado *HTTP* conteniendo el estado acompañando al cuerpo (*body*) en el response.

[15] En el anexo 1.6 del presente anexo se muestra la secuencia gráfica de la actualización de un API.
[16] https://semver.org/

c. Los objetos únicos deben ser retornados bajo la etiqueta *data* y no en el nivel superior del *JSON* de respuesta.

Tabla Nº 5: Ejemplo de adopción en las respuestas

PARA ADOPTAR	PARA NO ADOPTAR

Status: 200
Content-Type: application/json
{
“success”: true,
“code”: ”APP-0000”,
“messages”: [],
“data”:
{
"id": 123456,
"nombres": "Eliana",
"apellidoPaterno": "Quezada",
"genero": "Female",
"fechaIgreso": "2012-01-01",
"fechaNacimiento": "1990-01-01",
"oficina": "OTIC"
}
}
	Status: 200
	Content-Type: application/json
	{
	"id":123456,
	"nombres": "Eliana",
	"apellidoPaterno": "Quezada",
	"genero": "Female",
	"fechaIgreso": "2012-01-01",
	"fechaNacimiento": "1990-01-01",
	"odicina": "OTIC"
	}

d. En las etapas iniciales a la creación de las *API* se debe especificar los campos que posean información clasificada como confidencial (de acuerdo a la normativa vigente), estos campos deben cumplir con poseer controles de seguridad, que den tratamiento a los riesgos de seguridad que posean dichos campos.

e. Los errores deben estar referidos a los estados *HTTP* y deben manejarse con etiquetas *code* (con valor siempre en mayúsculas en las siglas de la aplicación), *messages* (con valor indicando la causa del error) y *errors* (que es opcional, contiene una lista de errores”). Por ejemplo:

En el caso de consultas:
Status: 404
Content-Type: application/json
{
“success”: false
"code": "APP-9000",
“error”: “Elemento no encontrado”
"messages": ["El empleado 1234456 no se encuentra registrado"]
}

En el caso de validaciones: Status: 404 Content-Type: application/json { “success”: false, "code": "APP-9001", "messages": ["La información suministrada no es válida."], "validations": [{ "message": "El apellido es requerido.", "field": "apellidoPaterno"

```
    },
    {
      "messages": ["La edad del empleado debe ser mayor que 18 años"],
      "field": "fechaNacimiento"
    }
  ]
}
```

```
Status: 401
Content-Type: application/json
{
  “success”: false,
  "code": "APP-9004",
  "messages": ["El acceso al sistema de empleados no ha sido otorgado."]
}
```

f. Para respuestas que contienen listas o arreglos de objetos como resultados de filtros de búsquedas se debe utilizar la alternativa que retorna los datos, los códigos de estado y los contadores en la cabecera de la respuesta.

```
Status: 200
Total-Count: 100
Content-Type: application/json
{
   “success”: true,
   “code”: ”0000”,
   “messages”: [],
   “data”:
   [
    {
     "employeeId":123456,
     "firstName": "Francisco",
     "lastName": "Medina",
     "gender": "Male",
     "hireDate": "2012-01-01",
     "birthDate": "1990-01-01",
     "office": "OTIC"
    },
    {
     "employeeId":123457,
     "firstName": "Mariela",
     "lastName": "Rojas",
     "gender": "Female",
     "hireDate": "2012-01-01",
     "birthDate": "1990-01-01",
     "office": "OTIC"
    }
  ]
}
```

```
Status: 200
Total-Count: 0
Content-Type: application/json
[]
```

g. Por defecto, no se debe optar por el retorno de valores nulos o vacíos. Sin embargo, si se desea conocer explícitamente si el valor retornado es nulo, se debe adoptar el retorno valores nulos o vacíos.

Adoptar siempre que se requiera aplicar únicamente para propósitos representacionales	**Para adoptar por defecto: No aparecen los valores nulos, pero sí aparecen los valores vacíos.**	**Para adoptar por defecto: No aparecen ni nulos ni vacíos.**

{
“employeeId”: 12345,
“firstName”: “Mónica”,
“lastName”: “Rojas”,
“mobile”: null,
“addresses”: []
}
	{
	“employeeId”: 12345,
	“firstName”: “Mónica “,
	“lastName”: “Rojas “,
	“addresses”: []
	}
		{
		"employeeId": 12345,
		"firstName": "Mónica",
		"lastName": "Rojas"
		}

Para respuestas con *Date* y *DateTime*, se debe usar el formato *ISO 860117.* Por ejemplo:

```
{
   "createdBy": "123456",
   "createdAt": "2012-01-01T18:25:43.511Z",
   "createdBy": "123456",
   "createdAt": "2012-01-01T18:25:43.511Z",
}
```

1.4.11 OPERACIONES PERMITIDAS CON EL MÉTODO *GET*

a. **Filtrado**: Se basa en el uso de parámetros por cada campo que implemente un filtro siempre cuando el retorno es una colección. Por ejemplo:

/cuadernodecontrol.minedu.gob.pe/.../cuadernos?seccion=S3A
Filtra solo los cuadernos de control de los estudiantes de la sección de tercero A de secundaria.

b. **Búsqueda:** Es posible implementar las búsquedas como si fuera un recurso. Ya se ha mencionado que no se puede utilizar verbos en la *URI*, pero esta es una excepción.

/cuadernodecontrol.minedu.gob.pe/.../cuadernos?seccion=S3A
Filtra solo los cuadernos de control de los estudiantes de la sección de tercero A de secundaria.

c. **Ordenación**: De forma similar al filtrado, el parámetro genérico *sort* se debe usar para describir reglas de ordenamiento. Organiza el ordenamiento según la lista de campos separados por comas, aplicando el signo negativo (-) indicando orden descendente de ser necesario.

GET /avisos?sort=-prioridad
Ordena los avisos según la prioridad de forma descendente.
GET /avisos?sort=-prioridad, fecha-creacion
Ordena los avisos según la prioridad de forma descendente y por fecha de creación

[17] https://www.w3.org/TR/NOTE-datetime

1.5 CONSIDERACIONES PARA LA DOCUMENTACIÓN DE LA *API REST*

a. **A nivel de cada recurso**, los métodos *GET, POST, PUT, DELETE* pueden obviar una descripción, sin embargo, los métodos que incluyan lógica de negocio, obligatoriamente deben estar descritos.
b. **A nivel de aplicación o servicio**, se debe incluir la siguiente información:
   - A nivel general:
     - Nombre de la *API.*
     - Descripción de la *API.*
     - Información del contacto (nombres y correo electrónico).
   - En cada método se debe indicar:
     - Objetivo, de forma obligatoria cuando se trate de métodos de negocio (por ejemplo: aprobar, anular, matricular, etc.).
     - El detalle de los parámetros de entrada.
     - El detalle de todos los mensajes de salida.
     - El detalle de los métodos de encriptación para el aseguramiento de la seguridad de la información transmitida.

1.6 TIPOS DE PLUGINS DEL *API-GATEWAY*

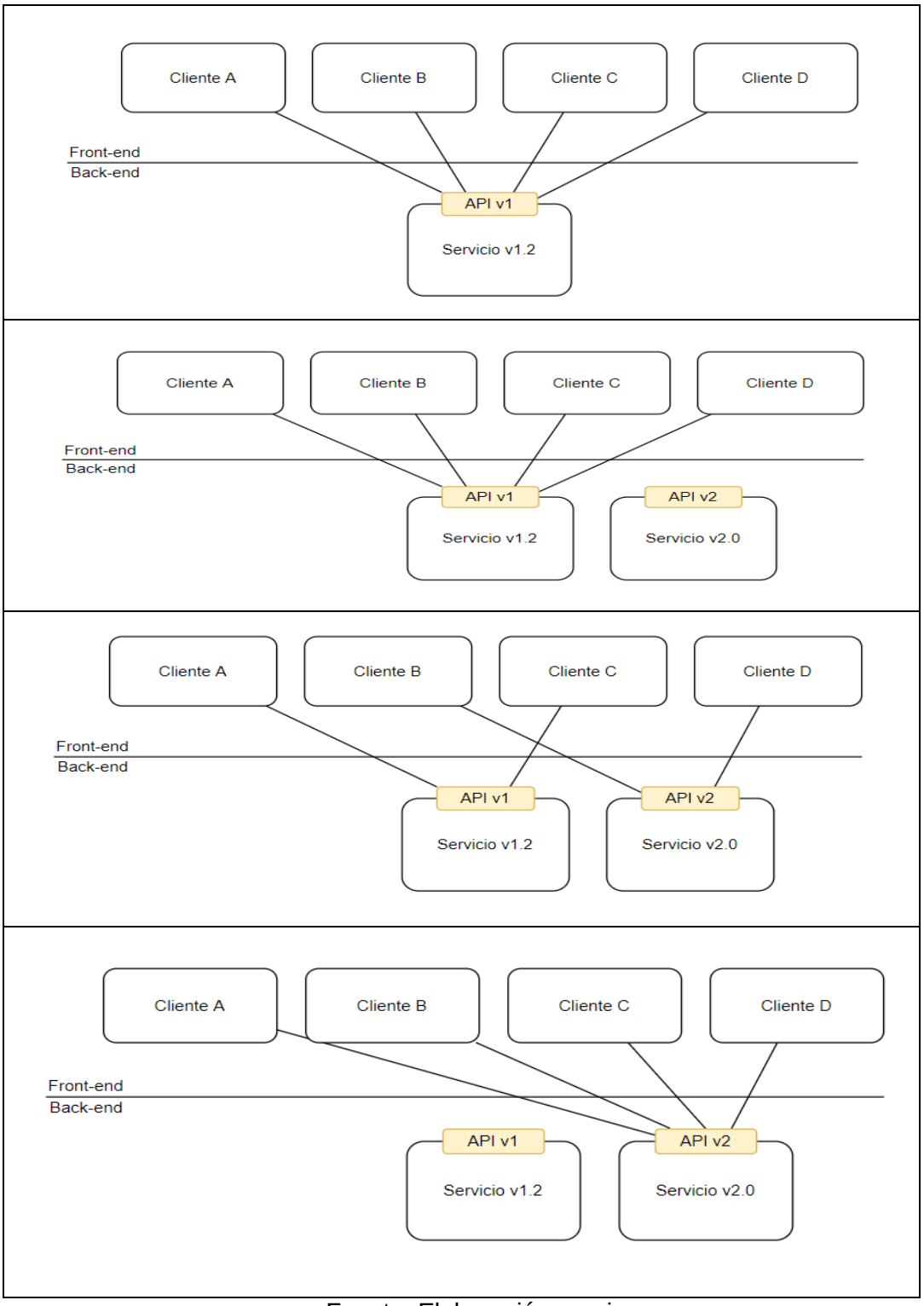
Tabla Nº 6: Tipos de Plugins para Implementar

Tipo	Adicional (*plugins*)	Descripción
Autenticación	JWT	Verifica peticiones conteniendo *Json Web Tokens*.
	OAuth2	Adiciona una capa con los flujos de *OAuth 2.0*. Ya que requiere el plugin de *SSL*, solo funciona con la versión empresarial de *Kong.*
Seguridad	Restricciones de IP	Maneja listas blancas y listas negras de direcciones *IP*.
	Detecciones de bot	Protege las *API* de los *bots* más comunes.
	ACL	Restringe el acceso a las *API* a través de las listas de control de acceso.
Control de tráfico	Ratios límite	Limita el número de peticiones *HTTP* por unidad de tiempo.
	Caché en *proxy*	Memoria intermedia para mejorar performance.
Analítica y monitoreo	Datadog	Instala agente *Datadog*
Logging	Http Log	Registra el *tracking* de las peticiones http configuradas.

Fuente: Elaboración propia

1.7 SECUENCIA GRÁFICA DE LA ACTUALIZACIÓN DE LA *API*

Gráfica Nº2: Secuencia de actualización de la *API*



Fuente: Elaboración propia

1.8 MECANISMOS DE SEGURIDAD DE LAS *API* PERMITIDOS

1.8.1 SEGURIDAD DE LAS *API*

- La adopción de mecanismos de seguridad debe proteger las *API* que sean desarrolladas y posteriormente implementadas.
- Para el caso de servicios que hagan uso de intercambios de credenciales (claves, llaves o señan de seguridad), se debe primar el uso de métodos de cifrados asimétricos.
- En el caso de usar métodos de cifrado simétricos se debe establecer un procedimiento de intercambio de dichas credenciales, denominado "ceremonia de claves".
- Es altamente recomendable seguir las buenas prácticas indicadas en el *OWASP API Security Project*.

1.8.2 MECANISMOS DE SEGURIDAD DE LAS *API*

a. ***HTTPS*:** Los servicios *API REST* en ambientes de producción deben ser expuestos solo a través de *endpoints HTTPS*. Con este mecanismo se protegen el contenido transmitido, entre ellos las credenciales de autenticación, contraseñas, identificadores de recursos o *JSON Web Tokens*.
   En los ambientes de desarrollo se debe implementar *HTTPS* a fin de poder realizar las pruebas correspondientes a la seguridad de las *API* desde el proceso de desarrollo.

b. ***JWT*:** Es una cadena de texto que tiene 3 partes codificadas en Base64, separadas por un punto (cabecera, cuerpo, firma) que generamos y es entregada a los clientes de una *API*. Se debe priorizar el uso en los *JWT* con algoritmo RS256.

c. **Límites de Tráfico (cuotas):** El acceso irrestricto no es una buena práctica para el *API Management* en los ambientes de producción, pues se contrapone a su escalabilidad y seguridad. Los límites de procesamiento a través de una *API* se miden en "transacciones por segundo" (TPS). Si se prevé que una *API* recibirá muchas solicitudes en algún momento de su uso, es posible aplicar un patrón de arquitectura denominado "*Throttling*" (estrangulamiento). Se debe tener en cuenta las siguientes estrategias:
   - **Limitar el número de peticiones:** Consiste en limitar el número de peticiones que un cliente puede realizar al servicio en un tiempo determinado, lo que obliga al sistema a medir el número de solicitudes por cliente, para finalmente denegar el servicio cuando el umbral ha sido alcanzado.
   - **Priorizar servicios:** Consiste en limitar o denegar los servicios no esenciales hasta que el sistema alcance su punto de normalización. Para implementar esta estrategia es importante poder identificar cuáles son los procesos prioritarios y cuáles son prescindibles, de tal forma que podamos apagarlos o degradarlos de forma rápida, evitando que el sistema llegue a un estado de saturación.
   - **Distribución priorizada de solicitudes:** Esta estrategia se utiliza el patrón multiinquilino (*multitenancy)*. A cada inquilino se le diferencia de otros según su importancia y se priorizan los mensajes mediante colas con diferentes prioridades. Esta estrategia puede ser usada en aplicaciones en la nube donde podemos distinguir inquilinos *VIP* de los no *VIP*.

   Consideraciones al implementar el patrón *Throttling*:

- La solución debe haber sido diseñada para soportar este patrón, pues resulta importante establecer la priorización de los servicios.
- La solución debe poder diferenciar entre un error de la aplicación y uno de la aplicación de este patrón, de esta forma el cliente podrá saber el motivo por el cual su solicitud está siendo rechazada.
- Este patrón se puede utilizar como una estrategia temporal mientras el escalamiento horizontal tiene lugar.

d. **Balanceo de carga y monitoreo de Servicios:** La utilización de microservicios representa mayores desafíos debido al mayor número de servicios a monitorear en disponibilidad y rendimiento. Por ello se debe aplicar estrategias que aseguren un adecuado balanceo de carga, que permitan el monitoreo de la salud de los servicios y del aislamiento de fallas, cuando los microservicios se despliegen en los ambientes de producción.

Tabla Nº 7: Estrategias de Despliegues de Servicio

ESTRATEGIA	TIPO	OBSERVACIÓN
Balance de carga	Por *DNS*, consiste en la configuración un dominio en el servidor *DNS* con direcciones IP de múltiples *hosts*, así las solicitudes de los clientes se distribuyen.	Se debe usar cuando los *hosts* sean de alto rendimiento o en la red interna de MINEDU.
	Por *round robin*, consiste en el envío de solicitudes a un servidor centralizado, que a su vez lo distribuye a los hosts en forma cíclica.	No se debe usar cuando se necesite sesiones consistentes.
	Por anillo, se mantienen sesiones consistentes entre clientes y *hosts* a través de un hash y cuando un *host* es removido, las sesiones se transfieren a un nuevo *host*.	El cliente también puede reiniciar la sesión cuando un host es removido.
Monitoreo de salud	Por monitoreo activo, el balanceador envía periódicamente una petición de chequeo a cada host. El host puede generar un t*imeout* o un código de error 500.	Su uso requiere elevado nivel de configuración (uno por cada comportamiento) y por lo tanto se introduce tráfico extra.
	Por monitoreo pasivo, el balanceador monitorea los errores mientras van ocurriendo, si éstos pasan un límite entonces se marca el *host* como no saludable.	Refiérase al documento “Marco de referencia para la implementación de *software*”.
Corto circuito	Cuando un host no está en buen estado, lo mejor es hacer que el tráfico fluya a otro host en buen estado.	Su uso aplica cuando un host viene recibiendo demasiadas solicitudes o deba reiniciarse.

Fuente: Elaboración propia

e. **Ofuscar**[18] **el identificador del recurso**: Las claves primarias de los registros en tablas de base de datos no deben ser utilizados como Id de los recursos, puesto que sería proclive a sufrir un ataque de seguridad mediante el cual se podría conocer cuál es el siguiente número de identificador de un recurso y recuperar u obtener toda la información asociada a dichos recursos (ataque de adivinación). Por ello se puede implementar el ofuscador como un servicio, como una función en la base de datos o una clase o como un *helper*.

Nota: Todas las variables que contienen información confidencial que son devueltas o remitidas a una *API* deben ser evaluadas por separado, respecto a los controles de seguridad que serán aplicados, en concordancia con la normativa vigente.

REPÚBLICA DEL PERÚ
FIRMA DIGITAL
MINISTERIO DE EDUCACIÓN

VILCHEZ INGA Cesar FAU 20131370998 hard

JEFE DE OTIC - OTIC MINEDU

Soy el autor del documento

2022/08/15 15:43:46

[18] Para la RAE ofuscar significa “oscurecer, turbar la vista”.