"Análisis para la reingeniería del Portal Paraguay"

Informe de Arquitectura

Portal Paraguay

Versión V1R1A01

Versión Software:	V1R1A01
Fecha Preparación	04-03-2024
Preparado Por:	GRUPO HORUS
Verificado Por:	Dirección Desarrollo – Dirección General Gobierno Digital

Histórico de Cambios del Documento

Versión	Fecha	Descripción	Autor
1.0	04-03-2024	Versión Inicial – Documento base preparado en el marco de la Orden de Trabajo Nro. 10	Grupo Horus Ing. Luis Cardozo

Tabla de contenido

1. Introducción 4
2. Objetivo General 4
3. Objetivo Específicos 4
4. Marco Legal 4
5. Glosario de Términos 5
6. Arquitectura 5
6.1. Proyectos backend y frontend 5
6.2. Api Gateway 5
6.3. Aplicación móvil 8
6.4. Pruebas unitarias 11
6.5. Librerías personalizadas en aplicaciones Javas 14
6.6. Observabilidad 15
6.7. Visión Lógica 17
6.8. Arquitectura por capas 19
6.9. Contenedores de aplicaciones 22
6.10. Orquestadores 23
6.11. Arquitectura de Despliegues 24
7. Roadmap 26
8. Conclusión 28

1. Introducción

En el desarrollo de aplicaciones web y móviles, la arquitectura de despliegues juega un papel fundamental para garantizar la eficiencia, la seguridad y la escalabilidad. Este documento presenta las recomendaciones para la arquitectura de las aplicaciones asociadas al Portal Paraguay, abarcando aspectos como el stack tecnológico, la orquestación de contenedores, la definición del lenguaje de programación y otras consideraciones importantes. El público objetivo está compuesto por arquitectos de software, desarrolladores, ingenieros de DevOps y jefes de proyecto.

2. Objetivo General

Brindar una guía clara y concisa para la implementación de las aplicaciones, asegurando que se cumplan con los requisitos de cada proyecto.

3. Objetivo Específicos

- Definir un stack tecnológico actualizado
- Analizar las ventajas y desventajas de Flutter y React Native.
- Analizar las ventajas y desventajas de Angular y ReactJS.
- Recomendar herramienta de orquestación de contenedores
- Implementar medidas de seguridad para proteger las aplicaciones.
- Listar recomendaciones de requerimientos para una mejora continua de la arquitectura

4. Marco Legal

- Ley 5282/2014 "De libre acceso ciudadano a la Información Pública y Transparencia Gubernamental"
- Decreto 8709/2018
- Ley Nro. 6207/2018
- Identidad Electrónica: 218/2020
- Portal Paraguay: https://www.paraguay.gov.py/

- Documentos disponibles: https://www.paraguay.gov.py/documentos
- Mi carpeta ciudadana: https://www.paraguay.gov.py/carpeta-ciudadana

5. Glosario de Términos

OEE: Organismo o Entidad del Estado

MITIC: Ministerio de Tecnologías de la Información y Comunicación

***Ciudadano**:* Usuario de la aplicación Portal Paraguay que genera un documento digital portable y un código QR del mismo para su uso en formato digital.

6. Arquitectura

6.1. Proyectos backend y frontend

Aplicaciones	Recomendaciones
Frontend	Actualizar a la última versión de Angular: • Angular v17.3
Backend	Actualizar a la última version de spring boot: • Spring Boot v3.2.4 Actualizar a la última version de Java: • OpenJDK 21 Migrar los proyectos legados a Spring boot.
Aplicación móvil	Actualizar a la última versión de Flutter

6.2. Api Gateway

API Gateway es un componente central en la arquitectura moderna de aplicaciones. Es un punto de entrada único y unificado para las APIs de una organización, lo que significa que los consumidores de APIs solo necesitan interactuar con la API Gateway para acceder a todas las APIs que necesitan.

Importancia de usar API Gateway:

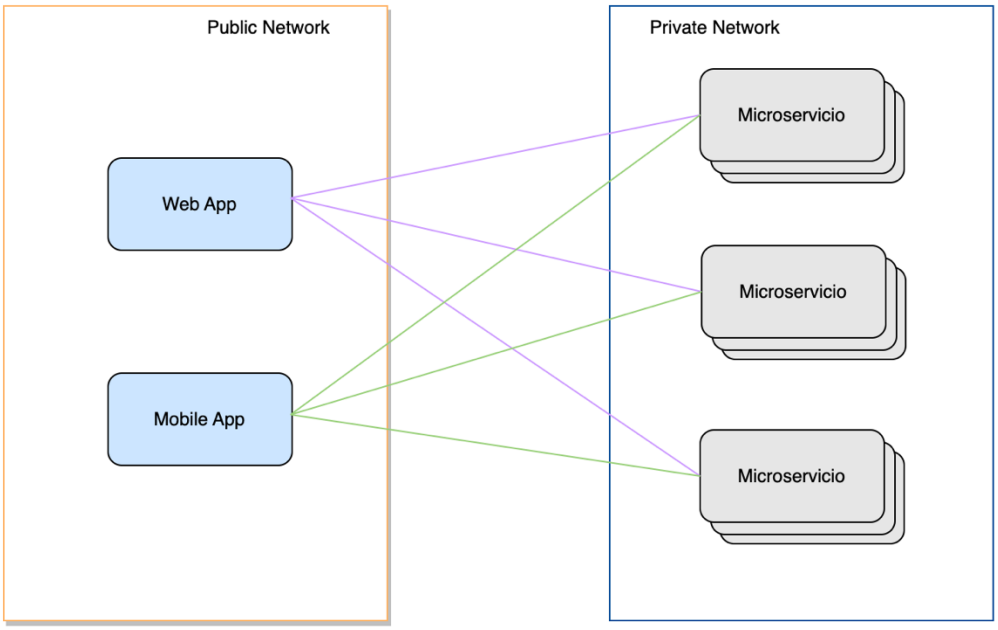
1. Seguridad: API Gateway ofrece una capa adicional de seguridad para las APIs, lo que ayuda a protegerlas de accesos no autorizados, ataques y vulnerabilidades.

2. Monitoreo y análisis: API Gateway permite monitorizar y analizar el tráfico de las APIs, lo que proporciona información valiosa sobre cómo se están utilizando las APIs y cómo se puede mejorar su rendimiento.

3. Escalabilidad: API Gateway puede escalarse para manejar grandes volúmenes de tráfico de APIs, lo que es esencial para las aplicaciones que necesitan soportar un alto número de usuarios o dispositivos.

4. Gestión del ciclo de vida de las APIs: API Gateway facilita la gestión del ciclo de vida de las APIs, desde su creación y publicación hasta su desactivación y eliminación.

5. Facilidad de uso: API Gateway simplifica el proceso de desarrollo e integración de APIs, lo que permite a los desarrolladores concentrarse en la lógica de negocio de sus aplicaciones.



Imágen 1. Api Gateway

En resumen: API Gateway es una herramienta esencial para las organizaciones que necesitan proteger, monitorizar, escalar y gestionar sus APIs. Su uso puede mejorar

significativamente la seguridad, el rendimiento y la facilidad de uso de las APIs, lo que se traduce en un mejor servicio para los usuarios y una mayor eficiencia para las empresas.

Comparativo de API Gateway

Wso2 API Manager:	**NGINX Plus:**
Ventajas	
Completo: Ofrece una gama completa de funciones de API Gateway, como seguridad, análisis, monetización y gestión del ciclo de vida de las API. **Flexible:** Se puede implementar en una variedad de entornos, incluyendo on-premise, en la nube y híbridos. **Escalable:** Puede manejar grandes volúmenes de tráfico de API. **Open Source:** La versión gratuita de WSO2 API Manager es de código abierto, lo que la convierte en una opción atractiva para organizaciones con presupuestos limitados.	**Fácil de usar:** NGINX Plus es relativamente fácil de configurar y gestionar. **Rendimiento:** NGINX Plus es conocido por su alto rendimiento y su capacidad para manejar grandes volúmenes de tráfico. **Costo:** NGINX Plus es más asequible que WSO2 API Manager, especialmente para organizaciones con presupuestos limitados.
Desventajas	
Complejo: La configuración y la gestión de WSO2 API Manager puede ser compleja, especialmente para organizaciones con recursos limitados. **Costoso:** La versión comercial de WSO2 API Manager puede ser costosa para organizaciones con presupuestos limitados.	**Funcionalidad:** NGINX Plus no ofrece la misma gama completa de funciones que WSO2 API Manager. **Escalabilidad:** NGINX Plus puede no ser tan escalable como WSO2 API Manager para organizaciones con grandes volúmenes de tráfico de API.

En resumen: WSO2 API Manager es una mejor opción para organizaciones que necesitan una solución completa de API Gateway con una amplia gama de funciones.

NGINX Plus es una mejor opción para organizaciones que necesitan una solución de API Gateway fácil de usar y asequible.

Consideraciones adicionales:

- **Requisitos específicos:** Es importante evaluar sus requisitos específicos de API Gateway antes de tomar una decisión.
- **Presupuesto:** El costo de la solución de API Gateway también es un factor importante a considerar.
- **Recursos:** Es importante tener en cuenta los recursos que tiene disponibles para configurar y gestionar la solución de API Gateway.

Recursos adicionales:

- WSO2 API Manager: https://wso2.com/products/api-manager/
- NGINX Plus: https://www.nginx.com/products/nginx/

6.3. Aplicación móvil

A continuación, se presenta una comparación detallada de ambos frameworks:

Flutter	**React Native**
Ventajas	
• **Rendimiento nativo:** Flutter compila el código Dart a código nativo, lo que proporciona un rendimiento similar al de las aplicaciones nativas. • **Interfaz de usuario:** Flutter ofrece una amplia gama de widgets personalizables para crear interfaces de usuario atractivas y receptivas.	• **Amplia comunidad:** React Native tiene una comunidad muy grande y activa, con muchos desarrolladores experimentados y empresas que lo utilizan. • **Flexibilidad:** React Native permite utilizar componentes nativos, lo que proporciona una mayor flexibilidad en el desarrollo de aplicaciones. • **Curva de aprendizaje:** React Native utiliza JavaScript, un

• **Desarrollo rápido:** Flutter permite un desarrollo rápido gracias a la recarga en caliente y la herramienta de diseño "hot reload". • **Comunidad en crecimiento:** La comunidad de Flutter está creciendo rápidamente y hay una gran cantidad de recursos disponibles en línea.	lenguaje de programación ampliamente conocido, lo que facilita su aprendizaje para los desarrolladores web. • **Amplio ecosistema de bibliotecas:** React Native tiene un amplio ecosistema de bibliotecas de terceros disponibles.
Desventajas	
• **Lenguaje de programación nuevo:** Dart no es tan conocido como JavaScript, lo que puede dificultar la búsqueda de desarrolladores con experiencia. • **Tamaño de la aplicación:** Las aplicaciones Flutter pueden ser más grandes que las aplicaciones nativas. • **Dependencia de Google:** Flutter es un framework desarrollado por Google, lo que puede generar cierta dependencia de esta empresa.	• **Rendimiento:** El rendimiento de React Native puede ser ligeramente inferior al de las aplicaciones nativas, especialmente en dispositivos más antiguos. • **Problemas de compatibilidad:** React Native puede tener problemas de compatibilidad con las últimas versiones de Android e iOS. • **Dependencia de componentes nativos:** El desarrollo de algunas funcionalidades puede requerir la creación de componentes nativos, lo que aumenta la complejidad del proyecto.
Cantidad de personas que usan	
Según la encuesta Stack Overflow Developer Survey 2023, Flutter es el 10º	Según la misma encuesta, React Native es el 5º framework web más popular y el 7º framework móvil más popular.

framework web más popular y el 15º framework móvil más popular.	
Escalabilidad	
Flutter es un framework altamente escalable, lo que lo hace adecuado para el desarrollo de aplicaciones grandes y complejas.	React Native también es un framework escalable, pero puede ser más desafiante escalar aplicaciones React Native muy grandes.
Versiones	
Flutter lanza nuevas versiones de forma regular, con nuevas características y mejoras.	React Native también lanza nuevas versiones de forma regular, pero con una frecuencia menor que Flutter.
Performance	
Flutter ofrece un rendimiento similar al de las aplicaciones nativas.	El rendimiento de React Native puede ser ligeramente inferior al de las aplicaciones nativas, especialmente en dispositivos más antiguos.
Recomendaciones de actualización de versiones	
Se recomienda actualizar a la última versión de Flutter siempre que sea posible, ya que las nuevas versiones incluyen mejoras en el rendimiento, correcciones de errores y nuevas características.	Se recomienda actualizar a la última versión de React Native solo si es necesario, ya que las actualizaciones pueden romper la compatibilidad con las versiones anteriores.

En resumen: Flutter y React Native son dos frameworks de desarrollo de aplicaciones móviles multiplataforma con sus propias ventajas y desventajas. La elección del framework adecuado dependerá de las necesidades específicas del proyecto, las habilidades del equipo de desarrollo y las preferencias personales.

Recursos adicionales:

- **Flutter:** https://flutter.dev/multi-platform/web
- **React Native:** https://reactnative.dev/

Además de la información proporcionada anteriormente, aquí hay algunos puntos adicionales a considerar:

- **Curva de aprendizaje:** Si el equipo de desarrollo ya está familiarizado con JavaScript, React

6.4. Pruebas unitarias

Las pruebas unitarias son un tipo de prueba de software que se utiliza para verificar el comportamiento de unidades individuales de código. En otras palabras, las pruebas unitarias se ejecutan para comprobar que cada módulo o función de una aplicación funciona como se espera.

Se listan algunos beneficios del desarrollo e integración de pruebas unitarias:

- **Mejora la calidad del código:** Las pruebas unitarias ayudan a detectar errores en las primeras etapas del desarrollo, lo que facilita su corrección y reduce el riesgo de que se propaguen a otras partes del código.
- **Mayor seguridad:** Las pruebas unitarias pueden detectar vulnerabilidades de seguridad en el código, lo que ayuda a proteger la aplicación de ataques.
- **Facilita el mantenimiento del código:** Las pruebas unitarias ayudan a comprender mejor el código y su comportamiento, lo que facilita su mantenimiento y actualización.
- **Aumenta la confianza en el código:** Las pruebas unitarias proporcionan una mayor confianza en la calidad y el comportamiento del código, lo que reduce el riesgo de errores en producción.

Herramientas para pruebas unitarias en Angular

Existen diversas herramientas para realizar pruebas unitarias en Angular. Algunas de las más populares son:

Herramienta	Descripción
Karma	• Es un framework de pruebas que permite ejecutar pruebas unitarias en el navegador. • Ofrece una amplia gama de funciones para escribir y ejecutar pruebas unitarias, como la posibilidad de ejecutar pruebas en paralelo, la cobertura de código y la generación de informes. • Es compatible con diferentes frameworks de pruebas, como Jasmine, Mocha y Jest.
Jasmine	• Es un framework de pruebas que permite escribir pruebas unitarias de forma legible y fácil de entender. • Ofrece una sintaxis sencilla para escribir pruebas unitarias, con funciones como describe, it y expect. • Es compatible con diferentes frameworks de pruebas, como Karma, Mocha y Jest.
Jest	• Es un framework de pruebas moderno que permite ejecutar pruebas unitarias de forma rápida y eficiente. • Ofrece una serie de funciones para escribir y ejecutar pruebas unitarias, como la posibilidad de ejecutar pruebas en paralelo, la cobertura de código y la generación de informes. • Es compatible con diferentes frameworks de pruebas, como Karma, Mocha y Jasmine.
Angular CLI	• La CLI de Angular proporciona una serie de comandos para realizar pruebas unitarias, como ng test y ng e2e. • Estos comandos permiten ejecutar pruebas unitarias en el navegador o en la línea de comandos. • También permiten generar informes de cobertura de código.

Herramientas para pruebas unitarias en Java:

Existen diversas herramientas para automatizar la ejecución de las pruebas unitarias en Java. Algunas de las más populares son:

Herramienta	Descripción
JUnit	es un framework de pruebas unitarias para Java que facilita la escritura y ejecución de pruebas para unidades de código individuales.
Mockito	es un framework de mocks para Java que permite crear objetos simulados para las pruebas unitarias.

Beneficios de usar JUnit y Mockito:

- **Detectar errores a tiempo:** Las pruebas unitarias ayudan a detectar errores en las primeras etapas del desarrollo, lo que facilita su corrección y reduce el costo de desarrollo.
- **Mejorar la calidad del código:** Las pruebas unitarias obligan a los desarrolladores a escribir código más limpio y modular, lo que mejora la calidad general del software.
- **Aumentar la confianza en el código:** Las pruebas unitarias proporcionan una mayor confianza en que el código funciona correctamente, lo que reduce el riesgo de errores en producción.
- **Facilitar el mantenimiento del código:** Las pruebas unitarias ayudan a comprender mejor el código, lo que facilita su mantenimiento y actualización.

```java
@Test
public void testSum() {
  // Crear un objeto simulado de la clase Calculator
  Calculator calculator = Mockito.mock(Calculator.class);

  // Cuando se llame al método sum() del objeto simulado, devolver 10
  Mockito.when(calculator.sum(2, 8)).thenReturn(10);

  // Invocar el método sum() del objeto simulado
  int result = calculator.sum(2, 8);

  // Verificar que el resultado es 10
  assertEquals(10, result);
}
```

Imágen 2. Ejemplo de prueba unitaria con Java

Integración con el flujo de desarrollo:

Las pruebas unitarias deben integrarse al flujo de desarrollo del proyecto. Se recomienda ejecutar las pruebas unitarias de forma automática antes de cada commit de código para asegurar que el código no se ha roto.

En resumen: las pruebas unitarias son una herramienta esencial para el desarrollo de software. Ayudan a mejorar la calidad del código, la seguridad, la facilidad de mantenimiento y la confianza en el código.

6.5. Librerías personalizadas en aplicaciones Javas

Las librerías personalizadas son una excelente manera de evitar la duplicación de código en aplicaciones Java, especialmente en arquitecturas de microservicios. Al crear una librería que encapsula funcionalidad común, como el control de sesiones con Redis, puedes reutilizarla en diferentes microservicios, simplificando el desarrollo y el mantenimiento.

Beneficios de usar librerías personalizadas:

- **Reducción de la duplicación de código:** Evita escribir el mismo código en cada microservicio que necesita la funcionalidad.
- **Mejora en la mantenibilidad:** Facilita la actualización y el mantenimiento del código al centralizarlo en una única librería.
- **Aumento de la modularidad:** Permite crear aplicaciones más modulares y reutilizables.
- **Mejora en la calidad del código:** Promueve la escritura de código limpio y bien organizado.

Ejemplo de uso de una librería personalizada para el control de sesiones con Redis:

- **Crear la librería:** Desarrolla una librería Java que encapsule la lógica para interactuar con Redis y gestionar las sesiones de usuario. La librería debe proporcionar métodos para iniciar, obtener, invalidar y actualizar las sesiones.

- **Dependencia en los microservicios:** Agrega la librería como dependencia en los proyectos de los microservicios que necesitan controlar las sesiones con Redis.
- **Utilizar la librería en los microservicios:** Implementa la lógica de control de sesiones en cada microservicio utilizando la librería.

Interoperabilidad entre microservicios:

Para que los microservicios puedan interoperar con la librería personalizada, es importante definir una interfaz pública bien documentada. La interfaz debe exponer los métodos necesarios para gestionar las sesiones de usuario de forma que los microservicios puedan interactuar con ella de forma transparente.

Recomendaciones:

- Diseñar la librería de forma modular y flexible para que pueda ser utilizada en diferentes tipos de aplicaciones.
- Documentar la interfaz pública de la librería de forma clara y precisa.
- Implementar pruebas unitarias para garantizar el correcto funcionamiento de la librería.
- Seguir las buenas prácticas de desarrollo de software al crear la librería.

En resumen: el uso de librerías personalizadas es una práctica recomendable para evitar la duplicación de código y mejorar la mantenibilidad de las aplicaciones Java, especialmente en arquitecturas de microservicios. La creación de una librería que encapsula funcionalidad común, como el control de sesiones con Redis, permite reutilizarla en diferentes microservicios, simplificando el desarrollo y la interoperabilidad.

6.6. Observabilidad

Las herramientas de monitoreo como ELK (Elasticsearch, Logstash y Kibana) y Graylog son esenciales para la gestión eficiente de logs y eventos en entornos de TI modernos. Estas herramientas ofrecen una serie de beneficios que pueden mejorar significativamente la observabilidad, la seguridad y el rendimiento de las aplicaciones y la infraestructura.

Beneficios de usar una herramienta de monitoreo:

- **Centralización de logs:** Permite centralizar todos los logs de diferentes fuentes (aplicaciones, servidores, dispositivos de red, etc.) en un único lugar, lo que facilita su análisis y gestión.
- **Visibilidad y análisis:** Permite obtener una vista completa y en tiempo real de todos los eventos que ocurren en el sistema, lo que facilita la identificación de problemas y la detección de anomalías.
- **Correlación de eventos:** Permite correlacionar eventos de diferentes fuentes para obtener una comprensión más completa de las causas de un problema.
- **Búsqueda y análisis históricos:** Permite realizar búsquedas y análisis de logs históricos para identificar tendencias y patrones, lo que puede ayudar a prevenir problemas futuros.
- **Detección de amenazas:** Permite detectar amenazas de seguridad como intrusiones, malware y ataques DDoS mediante el análisis de logs y eventos.
- **Notificaciones y alertas:** Permite configurar notificaciones y alertas para que los equipos de TI sean notificados de inmediato cuando se produzcan eventos críticos.
- **Cumplimiento normativo:** Permite cumplir con las normas y regulaciones que exigen la auditoría y el análisis de logs.

Ejemplos de herramientas de monitoreo:

- **ELK (Elasticsearch, Logstash y Kibana):** Es una suite de código abierto que ofrece una solución completa para la gestión de logs.
- **Graylog:** Es una herramienta de monitoreo comercial que ofrece una amplia gama de funciones para la gestión de logs y eventos.

Recomendaciones:

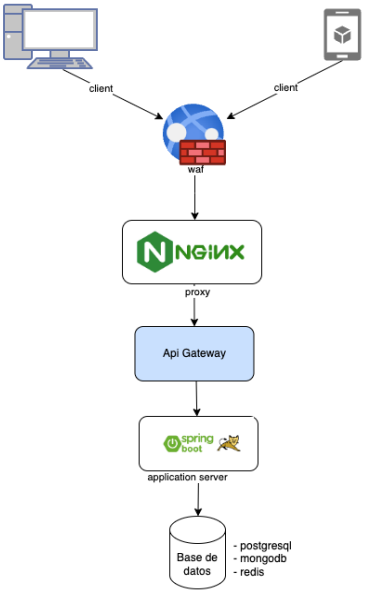
- Elegir la herramienta de monitoreo adecuada para las necesidades específicas.
- Implementar la herramienta de monitoreo de forma correcta y configurarla para que recopile los logs y eventos relevantes.
- Capacitar en el uso de la herramienta de monitoreo.

- Monitorizar los logs y eventos de forma regular para identificar y solucionar problemas.

En resumen: las herramientas de monitoreo como ELK o Graylog son una herramienta esencial para la gestión eficiente de logs. Estas herramientas pueden mejorar significativamente la observabilidad, la seguridad y el rendimiento de las aplicaciones y la infraestructura.

6.7. Visión Lógica

Se presenta una vista lógica general



Imágen 3. Visión lógica

Clientes: Los clientes son los usuarios finales del sistema, que pueden interactuar con él a través de diferentes interfaces como aplicaciones web, móviles o interfaces de línea de comandos.

WAF (Web Application Firewall): El WAF es un componente de seguridad que protege al sistema de ataques web comunes como SQL injection, cross-site scripting y ataques de denegación de servicio.

API Gateway: La API Gateway es un componente que actúa como puerta de entrada a los microservicios del sistema. Se encarga de recibir las solicitudes de los clientes, enrutarlas al microservicio adecuado y devolver las respuestas al cliente.

Microservicios: Los microservicios son unidades independientes de software que implementan funcionalidades específicas del sistema. Se comunican entre sí a través de una red interna y son responsables de la lógica de negocio del sistema.

Base de datos: La base de datos es un componente que almacena los datos del sistema. Puede ser una base de datos relacional, NoSQL o una combinación de ambas.

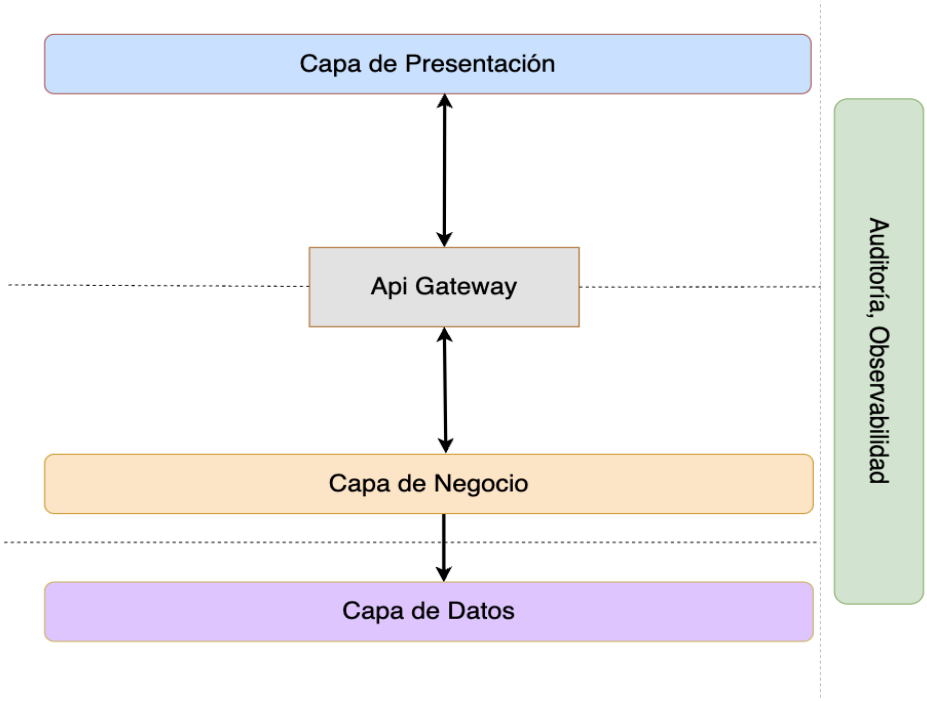
Flujo de información:

Las flechas en el diagrama indican el flujo de información entre los componentes. La información fluye desde los clientes al WAF, luego a la API Gateway y finalmente a los microservicios. Los microservicios pueden acceder a la base de datos para leer y escribir datos.

En resumen: la visión lógica del sistema muestra cómo los diferentes componentes interactúan entre sí para proporcionar la funcionalidad deseada al usuario final.

6.8. Arquitectura por capas

En una arquitectura de software por capas, se dividen las funcionalidades del sistema en diferentes capas, cada una con una responsabilidad específica. Esto ayuda a mejorar la organización, la modularidad y la mantenibilidad del código.



Imágen 4. Esquema General

Esquema General – Arquitectura por capas

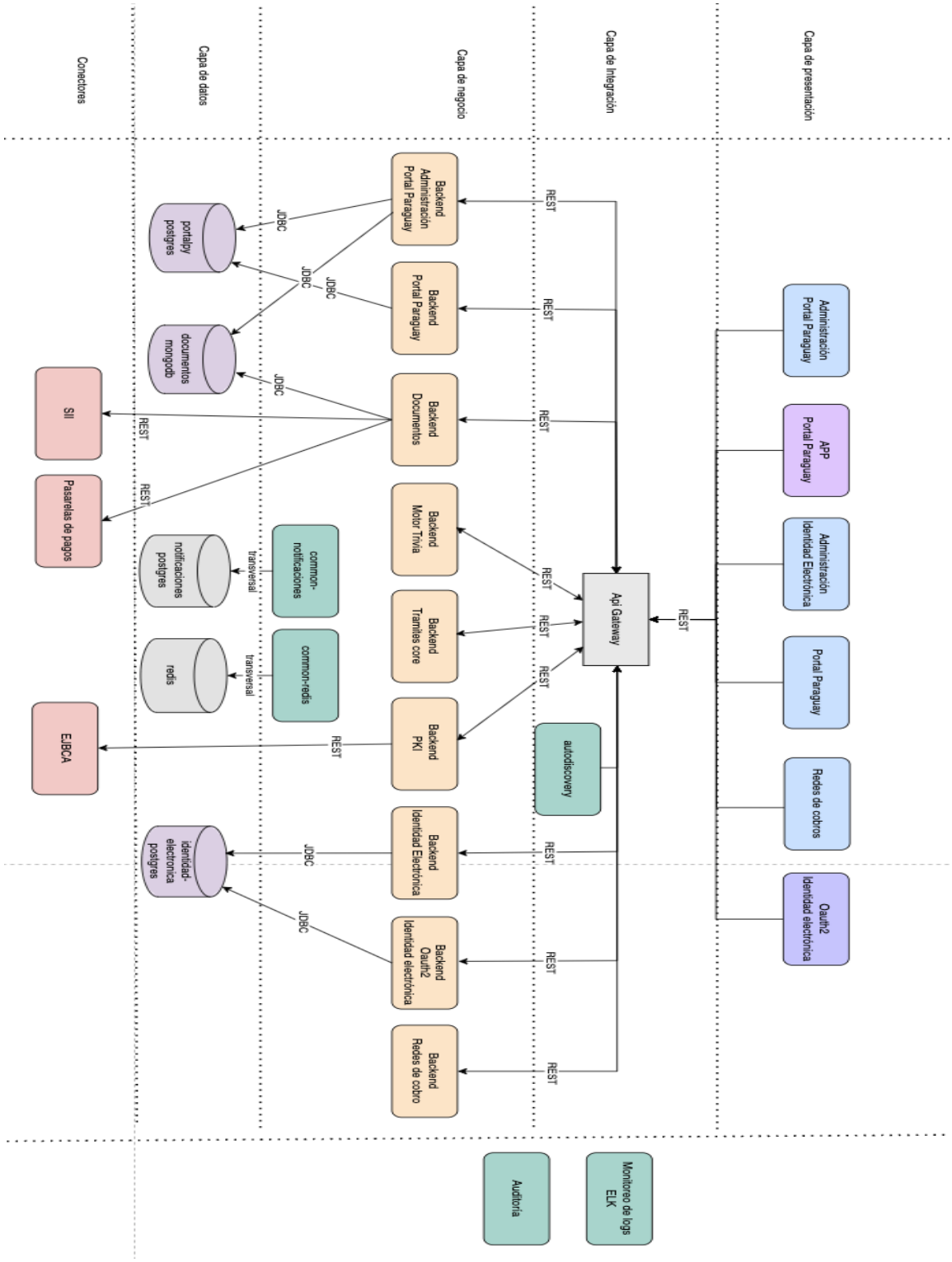
Capa de presentación	• **Responsabilidad:** Interactuar con el usuario final, mostrando información y recibiendo entradas. • **Componentes:** Interfaz de usuario (UI), controladores de presentación. • **Tecnologías:** HTML, CSS, JavaScript, frameworks web (React, Angular, Vue.js).
Capa de integración (API Gateway)	• **Responsabilidad:** Exponer una interfaz única para la interacción con los microservicios del sistema. • **Componentes:** API Gateway, controladores de API.

	• **Tecnologías:** WSO2, Nginx Plus.
Capa de negocio	• **Responsabilidad:** Implementar la lógica del sistema, procesando las reglas de negocio y realizando operaciones sobre los datos. • **Componentes:** Microservicios, controladores de negocio, lógica de negocio. • **Tecnologías:** Java, frameworks de desarrollo web (Spring boot).
Capa de datos	• **Responsabilidad:** Almacenar y recuperar datos del sistema. • **Componentes:** Base de datos, gestor de base de datos. • **Tecnologías:** PostgreSQL, MongoDB.
Capa transversal	• **Responsabilidad:** Proporcionar funcionalidades comunes a todas las capas del sistema, como auditoría, notificaciones y observabilidad. • **Componentes:** Módulos de auditoría, de notificaciones, de observabilidad. • **Tecnologías:** Elasticsearch, Kibana, Grafana, Graylogs.

Ventajas de una arquitectura por capas:

- **Modularidad:** Permite dividir el sistema en módulos independientes y reutilizables.
- **Mantenibilidad:** Facilita la identificación y corrección de errores.
- **Escalabilidad:** Permite escalar el sistema horizontalmente añadiendo más servidores a una capa específica.
- **Testabilidad:** Facilita la realización de pruebas unitarias y de integración.

Propuesta de una arquitectura por capas



Imágen 5. Arquitectura por capas

6.9. Contenedores de aplicaciones

Un contenedor es un paquete ligero, portátil y autosuficiente que incluye todo lo necesario para ejecutar una aplicación: código, runtime, herramientas del sistema, bibliotecas y configuraciones. Se aísla del resto del sistema y funciona de forma independiente, como si estuviera en su propio entorno virtual.

Ventajas de usar contenedores:

- **Portabilidad:** Los contenedores se pueden ejecutar en cualquier sistema operativo que tenga un motor de contenedores compatible, como Docker.
- **Ligereza:** Los contenedores son mucho más ligeros que las máquinas virtuales, ya que no incluyen un sistema operativo completo.
- **Eficiencia:** Los contenedores comparten el kernel del sistema operativo, lo que los hace más eficientes en el uso de recursos.
- **Escalabilidad:** Los contenedores se pueden escalar fácilmente de forma horizontal, lo que significa que se pueden agregar más contenedores a medida que aumenta la demanda.
- **Aislamiento:** Los contenedores se aíslan entre sí y del sistema operativo host, lo que aumenta la seguridad y la confiabilidad.

En la actualidad uno de los más usados es Docker, una plataforma de código abierto que facilita la creación, el envío y la ejecución de aplicaciones en contenedores. Docker proporciona una serie de herramientas para crear imágenes de contenedores, que son plantillas ligeras y ejecutables que contienen todo lo necesario para ejecutar una aplicación.

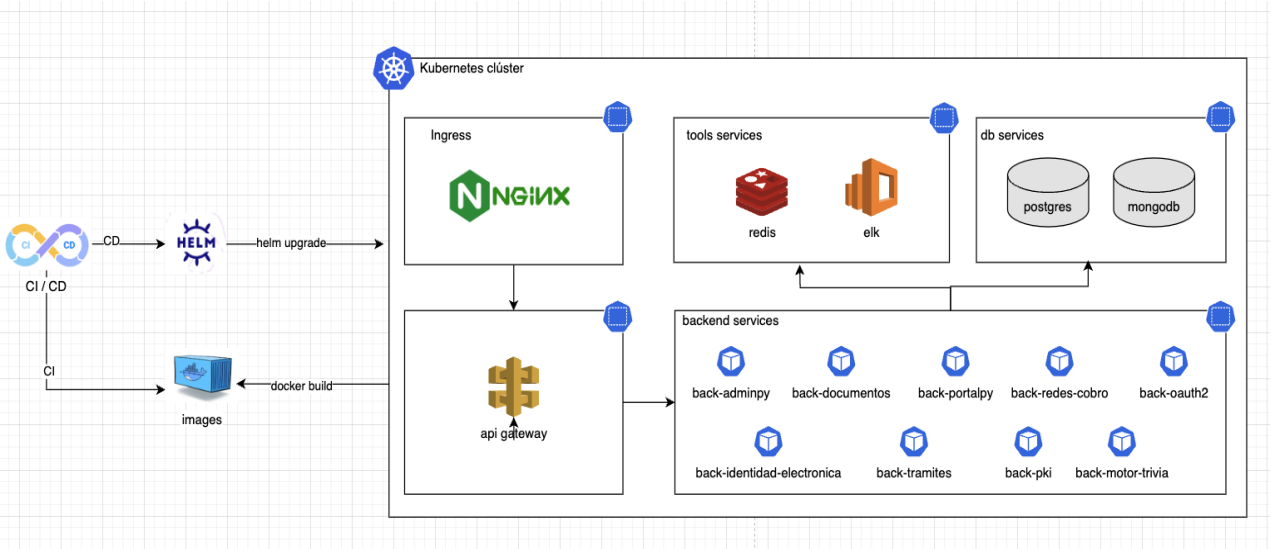
Ventajas de usar Docker con contenedores:

- **Facilidad de uso:** Docker proporciona una interfaz de línea de comandos simple y fácil de usar para trabajar con contenedores.
- **Automatización:** Docker puede automatizar la creación, el envío y la ejecución de contenedores.

- **Comunidad:** Docker tiene una gran comunidad de usuarios y desarrolladores que pueden ayudar a resolver problemas y encontrar soluciones.

6.10. Orquestadores

Los orquestadores de contenedores son herramientas que permiten gestionar y automatizar el ciclo de vida de los contenedores a gran escala. Algunos de los orquestadores de contenedores más populares son Kubernetes, Docker Swarm y Nomad.



Imágen 6. Propuesta del uso de orquestadores

Ventajas de usar un orquestador de contenedores:

- **Automatización:** Los orquestadores de contenedores pueden automatizar la programación, el escalado, el balanceo de carga y la recuperación ante fallos de los contenedores.
- **Escalabilidad:** Los orquestadores de contenedores permiten escalar las aplicaciones de forma horizontal de manera eficiente.
- **Alta disponibilidad:** Los orquestadores de contenedores pueden garantizar la alta disponibilidad de las aplicaciones.

En resumen: los contenedores son una tecnología poderosa para la entrega de aplicaciones. Docker facilita la creación, el envío y la ejecución de aplicaciones en contenedores. Los orquestadores de contenedores como Kubernetes permiten gestionar y automatizar el ciclo de vida de los contenedores a gran escala.

6.11. Arquitectura de Despliegues

La integración continua y la entrega continua (CICD) es un conjunto de prácticas que automatizan el proceso de desarrollo y entrega de software. Las herramientas de CICD ayudan a los equipos a desarrollar, probar e implementar software de manera más rápida y confiable.

Jenkins es una herramienta de automatización de código abierto que se utiliza para la integración continua y la entrega continua. Jenkins puede ejecutar una serie de tareas, como la compilación de código, la ejecución de pruebas y la implementación de software.

Nexus es un repositorio de artefactos que se utiliza para almacenar y administrar los artefactos de software, como bibliotecas, archivos JAR y Docker images. Nexus puede usarse para almacenar artefactos en diferentes formatos, como Maven, npm y Docker.

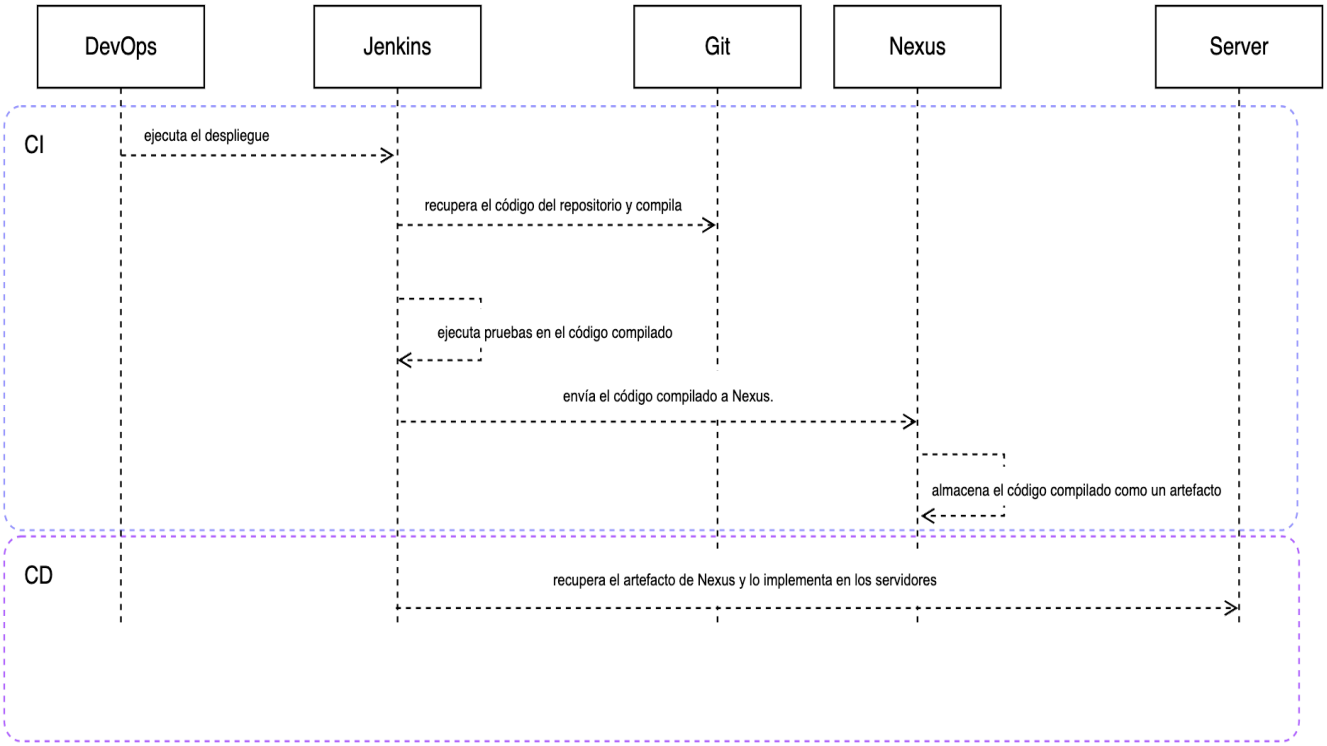
Spring Cloud Server es una plataforma de aplicaciones en la nube que se utiliza para construir y ejecutar aplicaciones en la nube. Spring Cloud Server proporciona una serie de servicios, como el descubrimiento de servicios, la балансировка de carga y la configuración de aplicaciones.

Beneficios de una arquitectura de despliegues CICD:

- **Automatización:** La arquitectura automatiza el proceso de desarrollo y entrega de software, lo que reduce el tiempo y el esfuerzo manual.
- **Confiabilidad:** La arquitectura mejora la confiabilidad del software al automatizar las pruebas y la implementación.
- **Velocidad:** La arquitectura permite a los equipos entregar software de manera más rápida y eficiente.
- **Escalabilidad:** La arquitectura es escalable y puede usarse para entregar software a una gran cantidad de usuarios.

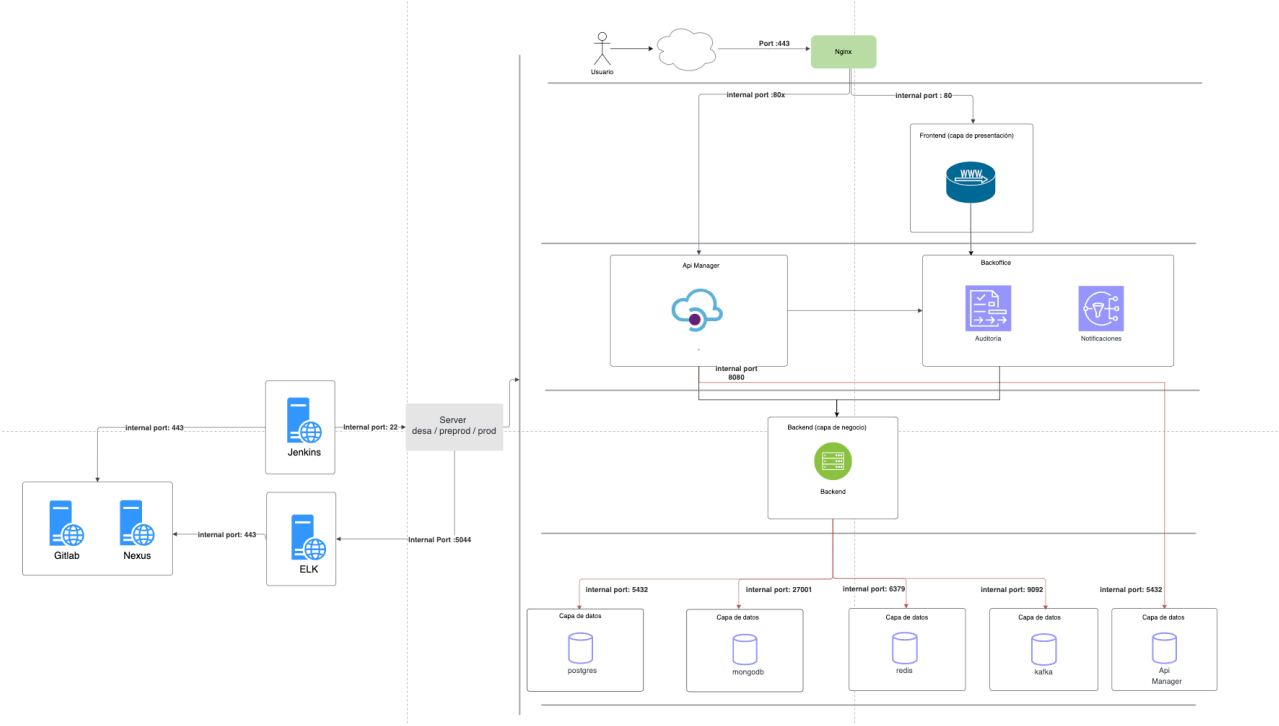
- **Seguridad:** La arquitectura puede usarse para mejorar la seguridad del software mediante la automatización de las pruebas de seguridad y la implementación de medidas de seguridad.

Ejemplo de secuencia de CI/CD:



Imágen 7. CICD de artefactos

Ejemplo de una arquitectura de despliegues:



Imágen 8. Arquitectura de Despliegues

7. Roadmap

Con el objetivo de mejorar la eficiencia, seguridad y escalabilidad de la infraestructura actual, se lista las recomendaciones a ser desarrolladas e implementadas.

DESCRIPCIÓN	PRIORIDAD
Actualización de tecnologías	
Actualizar versión de Java	ALTA
Actualizar versión de angular	ALTA
Migración de proyectos	
Migrar proyectos legados JavaEE a Spring Boot (versión actual)	BAJA
Desarrollo de librerías	
Desarrollar librería para el manejo de base de datos en caché REDIS	BAJA
Desarrollar librería para auditoría	BAJA
Desarrollar librería para notificaciones	BAJA

Pruebas Unitarias	
Desarrollar modelo de pruebas unitarias para aplicaciones Java a ser implementados en los nuevos proyectos	ALTA
Desarrollar modelo de pruebas unitarias para aplicaciones Angular a ser implementados en los nuevos proyectos	ALTA
Aplicaciones móviles	
Definir requerimientos funcionales para la aplicación móvil del Portal Paraguay	ALTA
Desarrollar aplicación móvil del Portal Paraguay	ALTA
Despliegues de aplicaciones	
Implementar Nexus para los paquetes compilados del frontend y backend	ALTA
Desarrollar pipeline y Job en Jenkins para CI/CD	ALTA
Dockerización	
Dockerizar las aplicaciones backend y frontend asociados al PortalPY	ALTA
API Gateway	
Implementar API Gateway	MEDIA
Observabilidad	
Implementar herramienta de monitoreo de logs	ALTA

Consideraciones:

- La hoja de ruta puede sufrir modificaciones a medida que avanza el desarrollo de los proyectos.
- El esfuerzo total estimado será revisado durante la etapa preparación de la OT, utilizando planilla de cálculo de esfuerzo.
- Es importante realizar un seguimiento constante del progreso de cada proyecto.
- Se recomienda la comunicación efectiva entre los diferentes equipos involucrados.

8. Conclusión

La implementación de las mejores prácticas descritas en este documento es fundamental para el desarrollo de software de alto rendimiento, seguro, escalable y sostenible a largo plazo. Estas prácticas no solo optimizan el proceso de desarrollo, sino que también garantizan la calidad, la confiabilidad y la eficiencia del software final, brindando un mejor servicio a los usuarios.

En resumen, las mejores prácticas para el desarrollo de software incluyen:

- Actualización constante de las versiones de Java y Angular: Garantiza la seguridad, el rendimiento y la compatibilidad con las últimas tecnologías.
- Implementación de pruebas unitarias exhaustivas: Mejora la calidad del código, la facilidad de mantenimiento y la confianza en el software.
- Creación y uso de librerías personalizadas: Evita la duplicación de código, facilita la mantenibilidad y la reutilización de funcionalidades.
- Adopción de contenedores y API Gateway: Permite la entrega eficiente, la escalabilidad y la seguridad de las aplicaciones.
- Utilización de herramientas de monitoreo como ELK o Graylog: Mejora la observabilidad, la seguridad y el rendimiento del software.

Recomendación:

- Integrar estas prácticas como pilares fundamentales del proceso de desarrollo de software.
- Capacitar al equipo de desarrollo en la comprensión y aplicación de estas mejores prácticas.

La adopción de estas mejores prácticas no solo conduce a la creación de software de alta calidad, sino que también fomenta una cultura de desarrollo ágil, eficiente y adaptable a las necesidades cambiantes del mercado. Invertir en la implementación de estas prácticas es una inversión estratégica que se traduce en un mayor valor para los usuarios, una mayor eficiencia y un éxito sostenido a largo plazo en el ámbito del desarrollo de software.

LUIS ALBERTO CARDOZO SERVIAN
Firmado digitalmente por LUIS ALBERTO CARDOZO SERVIAN
Fecha: 2024.03.25 08:34:23 -04'00'