Lineamientos Base de Arquitectura
Fábrica-Escuela de software
Programa de Ingeniería de sistemas

Propósito del documento
Este documento tiene como objetivo servir de guía inicial sobre lineamientos generales base de arquitectura que deben ser tenidos en cuenta por los equipos SCRUM en el proyecto asignado de la fábrica-escuela de software.

INFORMACIÓN DEL DOCUMENTO

Solicitado por:	Dirección de Fábrica-escuela CodeF@ctory UdeA		
Preparado por:	Sebastián Monsalve	**Revisado por:**	Diego Botia
Versión del documento:	1.0	**Fecha del documento:**	24/08/2023
Versión del documento:	2.0	**Fecha del documento:**	24/02/2024
Versión del documento:	3.0	**Fecha del documento:**	26/07/2024
Versión del documento:	4.0	**Fecha del documento:**	26/08/2025

HISTORIAL DE VERSIONES

Versión	Objetivo de la actualización	Fecha del documento	Autor
1.0	Elaborar primera versión para revisión	24/08/2023	Sebastián Monsalve
2.0	Elaborar segunda versión para revisión	24/02/2024	Diego Botia
2.0	Se incluye diagrama de arquitectura de seguridad y requerimientos no funcionales de seguridad	01/03/2024	Katerine Márceles
3.0	Se actualizan diagramas de arquitectura y elementos de las capas a implementar	26/07/2024 03/08/2024	Diego Botia Juan Pablo Arango
4.0	Revisión general para 2025-2	26/08/2025	Diego Botia Gustavo Palermo

Tabla de contenido

Alcance del documento

1. Arquitectura general de la solución

Dando cumplimiento a las diferentes consideraciones técnicas expresadas por el equipo de dirección de la fábrica-escuela CodeF@ctory UdeA, en los documentos de requerimientos, proponemos una arquitectura de solución que permitirá a la fábrica, no solo obtener la plataforma que se manifiesta en los requerimientos, sino que también podrán mantener los estándares definidos en cuanto al desarrollo de aplicaciones de software, tales como: escalabilidad, mantenibilidad, seguridad, usabilidad, protección de datos habeas data, internacionalización, trazabilidad.

Web Browser

Corresponde al cliente de la plataforma, donde se podrá visualizar y llevar a cabo todas las funcionalidades ofrecidas por la misma. Se recomienda el navegador Chrome para futuras pruebas automatizadas.

Identity and Access Management (IAM)

Es un componente centralizado que permitirá dar cumplimiento al requerimiento de acceso y gestión de permisos centralizado para los usuarios de la plataforma, tanto internos, como los externos.

Api Gateway

Es una capa entre el cliente y los recursos que ofrece la plataforma, sirviendo como único punto de entrada a los mismos, como beneficio dentro de la arquitectura se tiene:

- **Autenticación y autorización**: facilita el proceso centralizado de autenticación y autorización en el portal. Puede usarse JWT (JSON Web Tokens) para la gestión de sesiones de usuario. También, se puede revisar autenticación con OAuth 2.0 a través de plataformas como OKTTA.
- P**rocesamiento y validación de entradas:** facilita el pre-procesamiento de solicitudes antes de enviar la petición a los servicios.
- **Transformación de respuestas**: permite obtener respuestas diferenciadas para diferentes clientes a partir de la respuesta de un mismo servicio. Todas las respuestas deberán enviarse serializados a través del formato JSON.
- **Telemetría**: permite recolectar, administrar y visualizar de forma centralizada los logs del consumo de todos los servicios. Puede usar un sistema de gestión de logs como LogTash y Kibana. Además, podría usarse en el back Log4J que registra información importante como mensajes de error y entradas de un usuario en un programa.
- **Observabilidad**: Se creará un dashboard que permitirá visualizar en tiempo real el estado global de la aplicación usando Grafana y Prometheus.

- **Ofuscación de servicios:** permite desacoplar los servicios de los clientes, de esta forma los clientes acceden a una única url, la cual podrá ser respondida por servicios en la nube pública u On Premise, sin que el cliente final lo perciba. Además, facilita la escalabilidad y balanceo de cargas.

Frontend App

Corresponde al contenedor donde se va a desplegar la interfaz de usuario del portal. El desarrollo del frontend será mediante NEXT JS.

Backend API

Es el componente que alberga la aplicación que contiene toda la lógica de negocio del portal y expone servicios para ser consumidos por el Frontend, al igual que con el Frontend. La exposición de los servicios será a través de APIS creadas en REST o GraphQL.

API GraphQL:

Utiliza Spring Boot para crear una API basada en GraphQL que exponga los servicios necesarios para que el frontend interactúe con el backend de manera eficiente.

Diseñe endpoints con GraphQL que sigan las convenciones de nomenclatura y los métodos de Query y Mutaciones para operaciones facilitar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar). Todas las APIs deben estar versionadas. Se recomienda el uso del formato camelCase para el nombramiento de las variables y clases al interior del servicio

Resources Repository
Sirve como contendor de recursos usados por el portal, como imágenes, videos, documentos, entre otros.

Transactional Database

Es el repositorio para almacenar los datos de la aplicación, de acuerdo con las sugerencias en los estándares definidos. La base de datos será con el RDBMS PostgreSQL y será configurado con el servicio de nube Supabase.

Logging and Monitoring

Este componente centraliza los logs de los diferentes componentes de la infraestructura donde está alojada la plataforma, incluidos los generados por la misma, para supervisarlos, para que se puedan tomar medidas según alertas generadas.

Devops

Corresponde a los componentes para garantizar las operaciones de CI/CD. **Nota: Por favor leer el documento de lineamientos DevOps.**

Manejo de errores y excepciones:
Implementa un sistema de manejo de errores coherente en el backend para proporcionar respuestas claras y significativas en caso de fallos.

Utiliza códigos de estado HTTP adecuados y mensajes de error descriptivos para facilitar la depuración y el manejo de errores en el frontend.

Cache y optimización de consultas (Opcional):
Utiliza técnicas de caching para almacenar en memoria caché datos que se acceden y que no cambian con frecuencia.

Optimiza las consultas a la base de datos utilizando índices adecuados, consultas optimizadas y técnicas de paginación para mejorar el rendimiento y la escalabilidad del sistema.

Pruebas automatizadas:

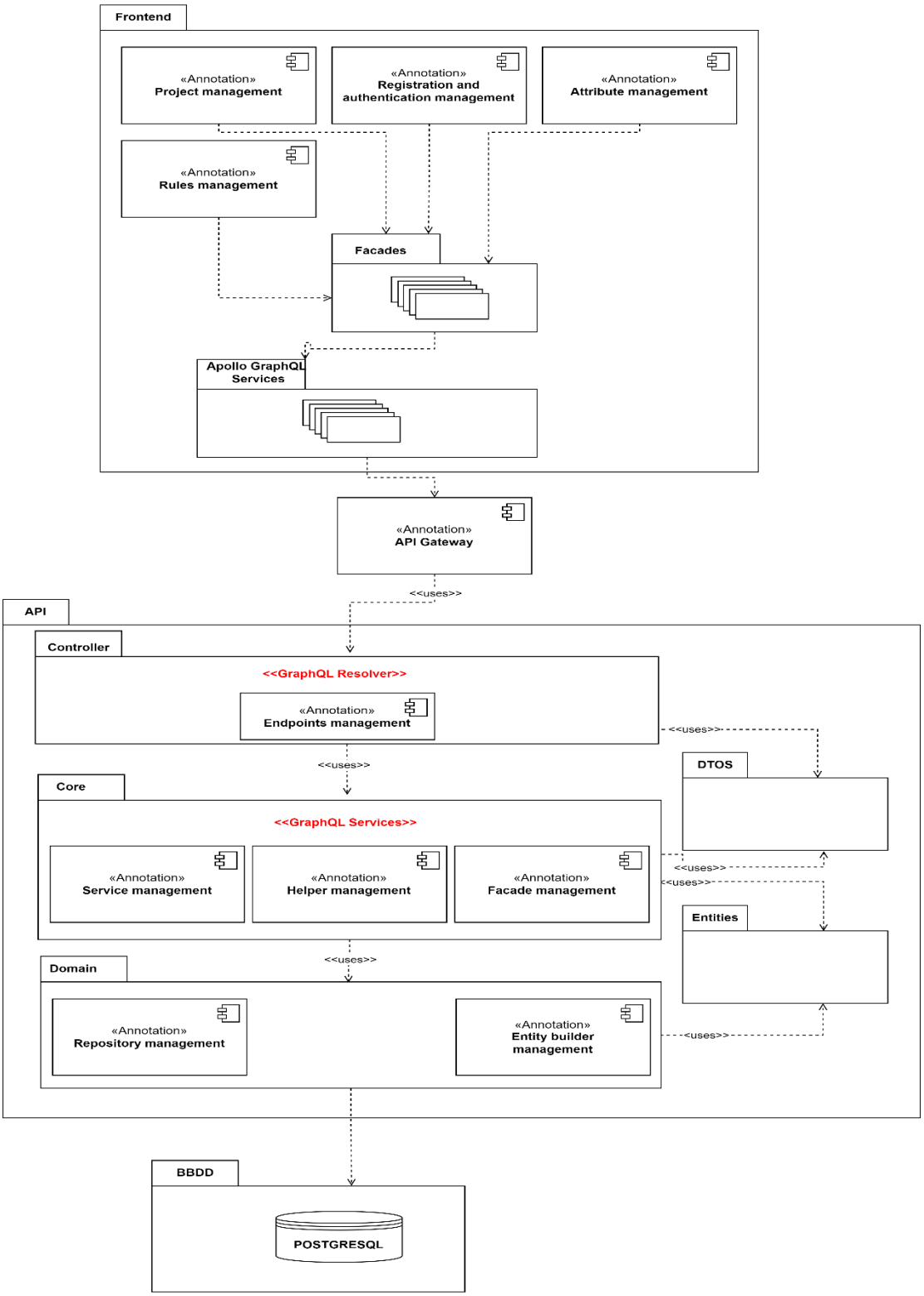
Escribir pruebas unitarias a cada miembro funcional de la aplicación y realizar las pruebas de aceptación en repo del producto.

Generar pruebas automatizadas funcionales validando transacciones completas de extremo a extremo mediante un repo independiente de automatización.

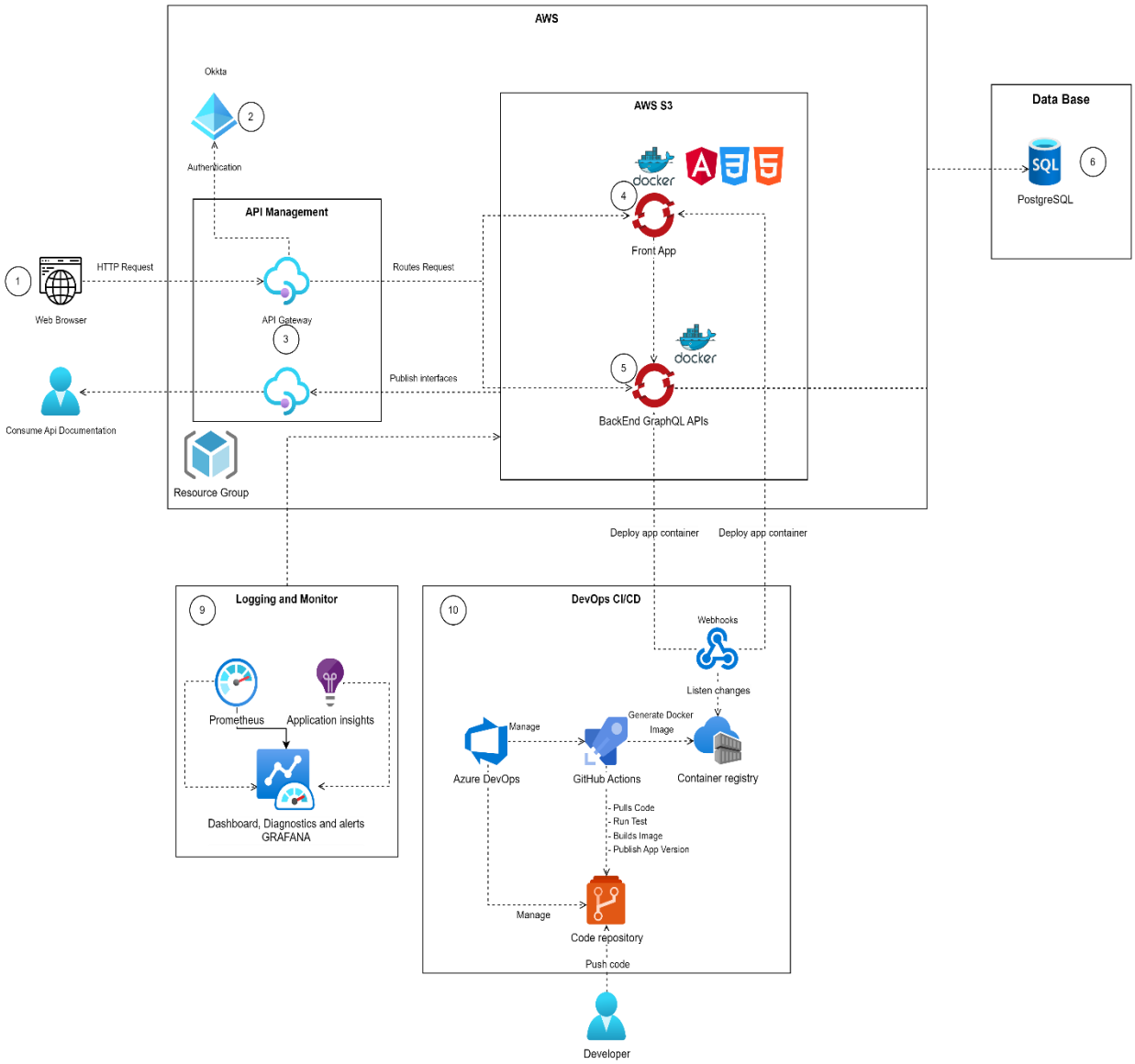
Utiliza herramientas de pruebas automatizadas E2E para ejecutar pruebas de manera regular y garantizar la estabilidad y confiabilidad del sistema.

2. Vista de arquitectura

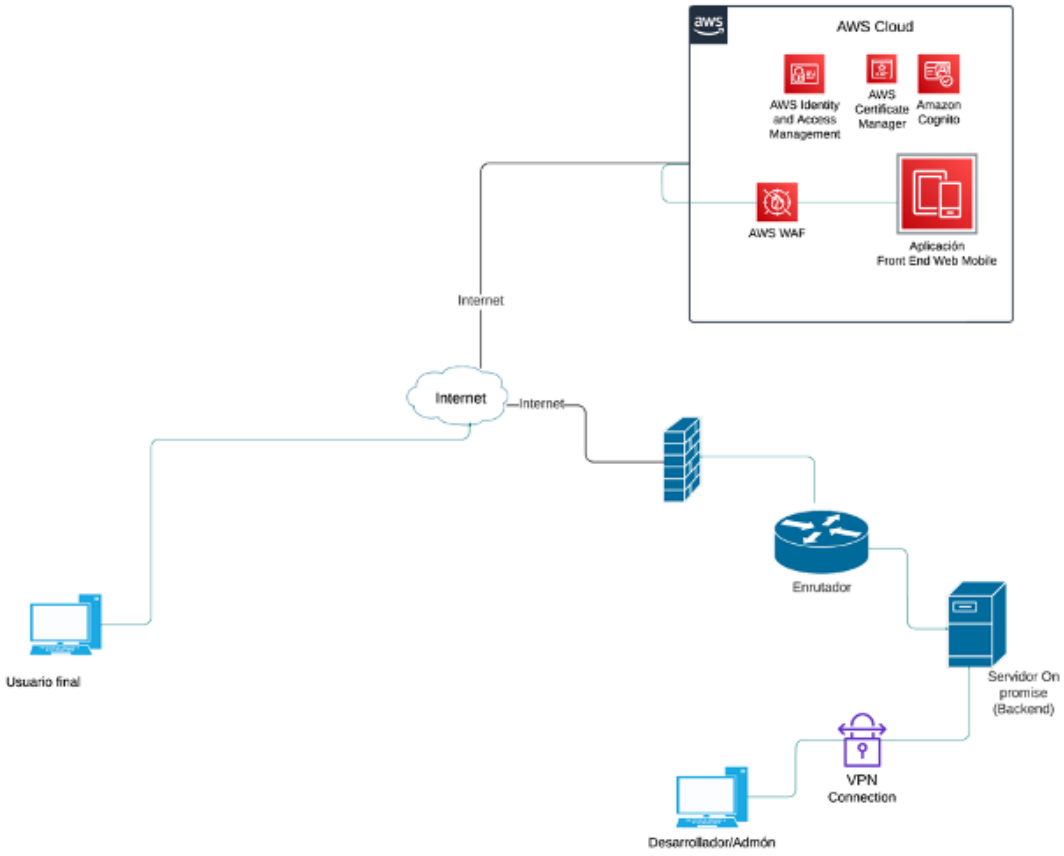
2.1 Vista de componentes



2.2 Vista de despliegue



2.3 Arquitectura de seguridad


AWS Cloud
AWS Identity and Access Management
AWS Certificate Manager
Amazon Cognito
AWS WAF
Aplicación Front End Web Mobile
Internet
Internet
Internet
Enrutador
Usuario final
Servidor On promise (Backend)
VPN Connection
Desarrollador/Admón

3. Lineamientos y principios de la arquitectura

3.1 Modelo Global por capas

Capa de Frontend:

- **Interfaz de Usuario (UI):**
  - La interfaz de usuario es la capa visible para el usuario final. En este caso, se implementa utilizando React, un framework de JavaScript ampliamente utilizado para la creación de interfaces de usuario interactivas y receptivas. Es posible emplear la librería NextJS para mejorar la funcionalidad de las interfaces.
  - La UI debería ser intuitiva y fácil de usar, con un diseño limpio y moderno que facilite la navegación y la interacción del usuario.
  - Utiliza componentes de React reutilizables para construir la interfaz de usuario, lo que promueve la modularidad y facilita el mantenimiento del código.

- **Gestión de Estado:**
  - React proporciona herramientas para gestionar el estado de la aplicación de manera eficiente. Puede utilizar Context API o bibliotecas de manejo de estado como Redux para administrar el estado global de la aplicación.
  - Centralice el estado de la aplicación y asegúrese de mantener un flujo de datos unidireccional para evitar problemas de sincronización y complejidad.
  - Se recomienda emplear React Hooks en lo posible.

- **Comunicación con el Backend:**
  - Utilice solicitudes HTTP para comunicarse con el backend y obtener o enviar datos. Puede hacer uso de la biblioteca Axios o Fetch API para realizar estas solicitudes.
  - Se emplearán servicios web basados en GraphQL empleando la librería Apollo o GraphiQL
  - Implemente manejo de errores y retroalimentación al usuario en caso de que ocurran problemas durante las solicitudes al backend.

- **Estilos y Diseño:**
  - Utilice CSS y/o preprocesadores como Sass para aplicar estilos a la interfaz de usuario. Considere el uso de bibliotecas de estilos como shadcn, tailwind, Bootstrap o Material-UI para facilitar el desarrollo y mantener un diseño consistente.
  - Diseñe la interfaz de usuario de manera responsiva para garantizar una experiencia de usuario óptima en diferentes dispositivos y tamaños de pantalla.

Capa de Backend:

- **Controladores/API:**
  - En el backend, Spring Boot proporciona un entorno de ejecución para aplicaciones Java.
  - Defina un controlador asociado a GraphQL utilizando las anotaciones que sean necesarias para exponer endpoints API que manejen las solicitudes HTTP entrantes.

- Implemente lógica de manejo de solicitudes en los controladores, procesando los datos recibidos, realizando operaciones en la base de datos y devolviendo las respuestas adecuadas al cliente.

- **Servicios:**
  - Separe la lógica de negocio de los controladores moviéndola a capas de servicios. Los servicios encapsulan la lógica de aplicación y promueven la reutilización y la modularidad del código.
  - Defina interfaces para los servicios y proporcione implementaciones concretas que realizan operaciones específicas, como la validación de datos, el procesamiento de lógica de negocio y la interacción con la capa de acceso a datos.

- **Acceso a Datos:**
  - Utilice Spring Data para interactuar con la base de datos PostgreSQL de manera eficiente. Spring Data proporciona abstracciones y herramientas para trabajar con diferentes tecnologías de persistencia de datos, incluida PostgreSQL.
  - Defina repositorios que extiendan las interfaces proporcionadas por Spring Data para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos.

- **Seguridad:**
  - Implemente mecanismos de seguridad para proteger los recursos del backend. Spring Security es una opción común para manejar la autenticación y la autorización en aplicaciones Spring Boot.
  - Configure filtros de seguridad para interceptar y validar solicitudes entrantes, autenticar usuarios y autorizar el acceso a recursos protegidos.
  - Puede usar herramientas de generación de tokens de sesión sobre las APIs como por ejemplo JWT o PASETO.
  - Debe verificar el cumplimiento de las pruebas de OWASP Top 10.

- **Gestión de Excepciones y Logging:**
  - Implemente un sistema de manejo de excepciones para capturar y gestionar errores de manera adecuada. Utilice anotaciones como **@ExceptionHandler** para manejar excepciones específicas y proporcionar respuestas de error significativas.
  - Integre un sistema de logging para registrar eventos importantes, errores y actividades del sistema. Utilice el marco de logging de Spring o bibliotecas como Log4j para esta funcionalidad.

3.2 Requisitos no funcionales

Escalabilidad: la solución debe ser fácilmente escalable de manera transparente y eficiente en todos sus componentes, se espera un crecimiento constante de usuarios de múltiples regiones del país y más adelante del mundo, y este crecimiento no debe afectar el desempeño y prestaciones ofrecidas por la solución con el tiempo. El escalamiento debe ser tanto horizontal como vertical y automático conforme aumenta o disminuye la demanda de las prestaciones ofrecidas por esta solución a sus clientes. Se recomienda usar el orquestador Kubernetes con Minikube.

Rendimiento: El sistema debe admitir una concurrencia de 200 solicitudes por minuto y cada solicitud debe responderse con un límite de 30 segundos.

Disponibilidad: la solución debe estar disponible, se espera un SLA de disponibilidad de todos los componentes de la solución de al menos 99.9%, un máximo de 8,76 horas de indisponibilidad al año. Hay que considerar que es una plataforma cuyas prestaciones podrían llegar a diferentes regiones del mundo en zonas horarias diferentes.

Mantenibilidad: debe ser fácil de mantener y evolucionar para agregar o quitar funcionalidades de acuerdo con nuevos requerimientos o necesidades que surjan en el tiempo. La solución debe ser modular en todas sus capas, con reglas de negocio parametrizables y estar soportada por servicios y/o componentes reutilizables pero independientes entre sí.

Interoperabilidad: debe ser fácilmente integrable con otras aplicaciones existentes o nuevas

Usabilidad: La plataforma debe ser intuitiva, fácil de manejar para los usuarios finales tanto internos como externos. El sistema debe garantizar que el portal proporcione una interacción positiva, intuitiva y satisfactoria, a través de un diseño atractivo, usabilidad fácil, rendimiento eficiente y atención a las emociones del usuario.

Requerimientos de seguridad

Entre los requisitos de seguridad a tener en cuenta para la construcción de la aplicación están:

Confidencialidad

- Cifrado de Datos: Implementar cifrado TLS para todas las transacciones en línea y cifrado AES para los datos almacenados, incluyendo información personal y de pago de los clientes.
- Acceso Seguro a la Base de Datos: Restringir el acceso a la base de datos con información sensible de clientes solo a usuarios autorizados usando listas de control de acceso y autenticación.

Integridad

- Firmas Digitales y Hashes: Utilizar firmas digitales para documentos electrónicos, como boletos y confirmaciones de reserva, y hashes para verificar la integridad de los datos transferidos.
- Control de Versiones de Software: Implementar un sistema de control de versiones para el código fuente que asegure la integridad del software utilizado en el sistema de reservas.

Disponibilidad

- Medidas Anti-DDoS: Adoptar servicios de mitigación de DDoS para proteger los servidores web y las infraestructuras críticas.
- Sistemas de Redundancia: Establecer sistemas redundantes y procedimientos de recuperación ante desastres para garantizar la disponibilidad continua del servicio de reservas.

Autenticación

- Autenticación Multifactor (MFA): Exigir MFA para todos los accesos al sistema de reservas por parte del personal y los clientes, combinando contraseñas con tokens o mensajes de texto.

- Políticas de Contraseñas Fuertes: Imponer políticas de contraseñas que requieran combinaciones complejas y renovaciones periódicas.
  Autorización
- Control de Acceso Basado en Roles (RBAC): Definir roles específicos dentro del sistema de reservas, asignando permisos precisos a cada rol para limitar el acceso a funciones y datos según la necesidad de conocer.

Auditoría y Monitoreo

- Gestión de Logs: Implementar un sistema centralizado de gestión de logs que registre todas las actividades críticas, incluidos accesos, transacciones y cambios en la configuración.
- Monitoreo de Eventos de Seguridad: Utilizar herramientas de detección de intrusiones y sistemas de gestión de eventos e información de seguridad (SIEM) para monitorear y alertar sobre actividades sospechosas en tiempo real.

Cumplimiento

- Cumplimiento con Regulaciones de protección de datos: Asegurar que todos los procesos y políticas de manejo de datos estén en conformidad con la Ley 1581 de 2012, así como con otras regulaciones locales e internacionales pertinentes a la protección de datos y la seguridad de la aviación.
- Auditorías de Seguridad: Realizar auditorías de seguridad regulares y evaluaciones de cumplimiento para identificar y corregir posibles deficiencias en las prácticas de seguridad.

3.3 Restricciones

3.4 Infraestructura y DevOps

El proyecto se desarrollará bajo prácticas de DevOps y marco de trabajo Scrum, lo que permite mejorar la comunicación entre los equipos (desarrollo y operaciones), obtener retroalimentaciones de cualquier actividad desarrollada y automatiza la mayor cantidad de actividades en el desarrollo de software posible, agilizando el proceso de entrega de funcionalidades al usuario final. El desarrollo será guiado por buenas prácticas de:

- Integración continua
- Despliegue/Entrega continua
- Evaluación de métricas de calidad (Análisis de código estático)
- Code review
- Automatización de pruebas (Unitarias, integración, funcionales) Desarrollo seguro

NOTA: Leer el documento de lineamientos de DevOps para ampliar esta información.