Tabla de Contenidos

1. INTRODUCCIÓN

NovaAir tiene como visión convertirse en una aerolínea innovadora y eficiente, destacándose por su tecnología avanzada en la gestión interna de vuelos y

operaciones. Nos proponemos facilitar la labor de los administradores a través de una plataforma intuitiva que permita gestionar de manera ágil y precisa todos los aspectos operativos de la aerolínea, desde la programación hasta la cancelación de vuelos. Nuestra misión es ofrecer una herramienta robusta, segura y eficiente, que optimice los procesos administrativos, permitiendo que los encargados de la aerolínea puedan ofrecer un servicio sin contratiempos, mejorando la operativa y la toma de decisiones en tiempo real.

Teniendo como iniciativa la implementación de soluciones tecnológicas avanzadas, buscamos simplificar y automatizar los procesos administrativos de la aerolínea. La plataforma que desarrollaremos permitirá a los administradores gestionar las reservas, programar vuelos, cancelar itinerarios y realizar otras tareas clave con facilidad. Además, garantizamos que todas las operaciones sean eficientes, seguras y escalables para acompañar el crecimiento de la aerolínea.

2. Problema

La gestión manual de vuelos y reservas puede resultar ineficiente y propensa a errores, lo que genera retrasos y dificultades para los administradores de aerolíneas. La falta de una interfaz unificada y fácil de usar para controlar las operaciones diarias puede afectar la puntualidad de los vuelos, la asignación de recursos y la toma de decisiones informadas en tiempo real. Los procesos desactualizados o poco automatizados también generan una carga administrativa innecesaria, lo que impacta negativamente en la eficiencia operativa de la aerolínea.

3. Soluciones

3.1. 1ra Solución: Desarrollar una plataforma administrativa que permita a los administradores gestionar de manera eficiente y centralizada todas las operaciones de la aerolínea. Esta plataforma incluirá funcionalidades para programar vuelos, realizar cambios o cancelaciones de vuelos, controlar el estado de las reservas, gestionar la ocupación de los aviones y emitir reportes detallados en tiempo real. La solución estará basada en una interfaz gráfica amigable que minimice el tiempo dedicado a tareas manuales, mejorando la eficiencia operativa y reduciendo el riesgo de errores.

3.2. 2da Solución: Implementar un sistema de **notificaciones automatizadas** para que los administradores reciban alertas sobre cualquier incidencia o cambio en la programación de vuelos, cancelaciones, modificaciones de rutas, o cualquier otra variable crítica. Esto permitirá una respuesta rápida y decisiones más informadas, optimizando la toma de decisiones y mejorando la coordinación interna de la aerolínea. La interfaz también contará con una vista en tiempo real de la disponibilidad de vuelos y recursos, permitiendo a los administradores realizar ajustes cuando sea necesario.

4. JUSTIFICACIÓN

La principal razón para llevar a cabo este proyecto es la necesidad de optimizar la administración interna de una aerolínea. Actualmente, las aerolíneas enfrentan desafíos al gestionar las operaciones de vuelo sin herramientas integradas que permitan realizar cambios y ajustes de manera eficiente. Con la creación de una interfaz gráfica intuitiva para administradores, buscamos simplificar y automatizar tareas complejas, como la programación de vuelos, la gestión de cancelaciones y la optimización de recursos. Esta plataforma no solo aumentará la eficiencia operativa, sino que también permitirá a los administradores tomar decisiones más informadas y mejorar la calidad del servicio ofrecido, reduciendo costos y errores operacionales.

5. Objetivos del Proyecto NovaAir

5.1. Objetivos primarios del Proyecto NovaAir

1. **Optimizar la gestión administrativa de la aerolínea:** Desarrollar una plataforma administrativa integral que permita a los administradores gestionar de manera eficiente todas las operaciones internas de la aerolínea, como la programación, cancelación y modificación de vuelos, con el fin de mejorar la eficiencia operativa y reducir los tiempos de gestión.
2. **Automatizar tareas clave:** Implementar funcionalidades automatizadas para procesos repetitivos, como la notificación de cambios en los vuelos, la actualización de la ocupación de los aviones y la gestión de reservas, para reducir errores humanos y agilizar la toma de decisiones.
3. **Facilitar la toma de decisiones en tiempo real:** Proveer a los administradores de herramientas que les permitan monitorear el estado de los vuelos, las reservas y otros aspectos operacionales en tiempo real, facilitando la toma de decisiones informadas y oportunas.
4. **Diseñar una interfaz gráfica intuitiva:** Crear una interfaz gráfica fácil de usar y accesible que permita a los administradores realizar las tareas operativas de manera rápida y sin complicaciones, minimizando la curva de aprendizaje y mejorando la productividad.
5. **Garantizar la seguridad y confiabilidad de las operaciones:** Asegurar que la plataforma sea segura, con accesos controlados para los administradores y otros usuarios, evitando posibles fallos o brechas de seguridad en la gestión de las operaciones y datos sensibles.
6. **Mejorar la coordinación interna de la aerolínea:** Facilitar la comunicación y coordinación entre los diferentes departamentos operativos, permitiendo que los administradores tengan una visión clara de todos los aspectos clave de las operaciones aéreas, optimizando los recursos y mejorando la eficiencia general.

5.2. Objetivos Secundarios del Proyecto NovaAir

1. **Mejorar la accesibilidad y usabilidad de la plataforma:** Diseñar la plataforma para que sea accesible desde diferentes dispositivos, como computadoras de escritorio, tabletas y móviles, garantizando que los administradores puedan gestionar las operaciones de la aerolínea desde cualquier lugar y en cualquier momento.
2. **Facilitar la formación y capacitación de los administradores:** Incluir tutoriales interactivos y materiales de capacitación dentro de la plataforma para que los administradores puedan aprender a utilizar las herramientas de manera rápida y eficiente, reduciendo el tiempo de adaptación y asegurando una transición suave al nuevo sistema.
3. **Promover la escalabilidad del sistema:** Diseñar la plataforma de manera modular y escalable, para que pueda crecer junto con la aerolínea y adaptarse a futuras necesidades operativas, como la incorporación de nuevas rutas, flotas o funciones administrativas adicionales.
4. **Mejorar el seguimiento de incidencias y problemas operativos:** Incluir un sistema de registro y seguimiento de incidencias o problemas operativos (por ejemplo, vuelos retrasados, mantenimiento de aviones), para que los administradores puedan gestionar y resolver situaciones de manera proactiva.
5. **Fomentar la seguridad de la información:** Asegurar que la plataforma cumpla con los más altos estándares de seguridad en la gestión de datos sensibles, protegiendo la privacidad y confidencialidad de la información tanto de la aerolínea como de los pasajeros.

6. Producto

El software está diseñado principalmente para gestionar de manera eficiente las operaciones administrativas de una aerolínea, brindando una experiencia de uso sencilla y segura para los administradores. Nuestra plataforma incluye funcionalidades para la programación de vuelos, cancelaciones y modificaciones, con un sistema robusto de gestión que asegura la correcta ejecución de cada operación.

El sistema de autenticación incluye procesos de registro e inicio de sesión, con la implementación de medidas de seguridad avanzadas para proteger la información sensible de la aerolínea y los pasajeros. Además, contamos con un módulo de gestión de vuelos que garantiza la correcta programación, modificación y

cancelación de los vuelos de manera estricta, para asegurar la eficiencia operativa de la aerolínea.

Por último, hemos desarrollado una interfaz gráfica intuitiva que facilita la administración de vuelos de manera ágil y precisa. El sistema proporciona un panel de control con informes y alertas en tiempo real, permitiendo a los administradores gestionar los vuelos, controlar recursos y asegurar que las operaciones diarias se realicen sin inconvenientes, proporcionando a los usuarios una plataforma confiable y eficaz para mantener el control total sobre las operaciones de la aerolínea.

6.1. Requerimientos Funcionales

1. **Gestión de vuelos**:
   El sistema debe permitir a los administradores programar, modificar y cancelar vuelos de manera eficiente. Esto incluye la capacidad de actualizar la información de los vuelos, como horarios, rutas, y disponibilidad.
2. **Gestión de recursos**:
   Los administradores deben poder gestionar los recursos de la aerolínea, como los aviones, el personal (pilotos, azafatas, personal de tierra) y otros elementos necesarios para las operaciones de vuelo.
3. **Control de seguridad**:
   El sistema debe implementar un sistema de autenticación y autorización para garantizar que solo los administradores autorizados puedan acceder a las funcionalidades del sistema. Además, se debe registrar un historial de todas las acciones realizadas por los administradores.
4. **Interfaz intuitiva**:
   La interfaz gráfica debe ser intuitiva y fácil de usar para que los administradores puedan realizar sus tareas de manera rápida y sin complicaciones. Esto incluye un diseño claro, accesible y con navegación eficiente.

6.2. Requerimientos No Funcionales

1. **Rendimiento**:
   El sistema debe ser capaz de gestionar grandes volúmenes de datos y realizar operaciones en tiempo real sin afectar la velocidad o el rendimiento del sistema. Las actualizaciones de los vuelos, reservas y recursos deben ser procesadas rápidamente.
2. **Escalabilidad**:
   El sistema debe ser escalable para poder soportar el crecimiento de la aerolínea, incluyendo la adición de nuevas rutas, vuelos, recursos, y la expansión a nuevas ubicaciones. Esto implica que el sistema debe permitir la incorporación de nuevas funcionalidades sin afectar la operatividad.
3. **Disponibilidad**:
   El sistema debe estar disponible para los administradores en todo momento, con una alta tasa de disponibilidad. Se debe minimizar el tiempo de inactividad y contar con medidas de respaldo en caso de fallos técnicos.
4. **Seguridad**:
   El sistema debe cumplir con los estándares de seguridad, protegiendo los datos sensibles relacionados con las operaciones aéreas y la información personal de los pasajeros. Esto incluye la implementación de cifrado de datos y protocolos de seguridad para prevenir accesos no autorizados.
5. **Compatibilidad**:
   La interfaz debe ser compatible con diferentes sistemas operativos (Windows, MacOS, Linux) y dispositivos (computadoras de escritorio, tabletas, teléfonos moviles, etc.), para garantizar que los administradores puedan acceder al sistema desde diversas plataformas.
6. **Usabilidad**:
   La interfaz debe ser fácil de usar, con una curva de aprendizaje mínima para los administradores. Debe ofrecer una experiencia de usuario fluida, con un diseño de interfaz atractivo, accesible y que minimice errores durante la interacción.
7. **Mantenimiento**:
   El sistema debe ser fácil de mantener y actualizar, permitiendo corregir errores y agregar nuevas funcionalidades sin interrumpir las operaciones diarias de la aerolínea.
8. **Integración**:
   El sistema debe ser capaz de integrarse con otros sistemas existentes en la aerolínea, como el sistema de gestión de reservas, el sistema de gestión de flotas y el sistema de mantenimiento de aeronaves, para asegurar una operación integrada y eficiente.

7. HISTORIAS DE USUARIOS

<table>
<tr><th colspan="4">HISTORIA DE USUARIO</th></tr>
<tr><td colspan="4">Prioridad: ALTA</td></tr>
<tr><td>Código del Requerimiento:</td><td>NP0001</td><td>Actor:</td><td>DLGG</td></tr>
<tr><td colspan="2">Nombre del Requerimiento:</td><td colspan="2">Creación del repositorio</td></tr>
<tr><td colspan="4">Descripción</td></tr>
<tr><td colspan="4">se requiere la creación de un repositorio central para el proyecto,
para que todos los integrantes del equipo puedan colaborar de manera eficiente, tener acceso a las actualizaciones en tiempo real y mantener el control de versiones.</td></tr>
<tr><td colspan="4">Funcionalidad</td></tr>
<tr><td colspan="4">Se debe incluir a todos los miembros del equipo de desarrollo para que puedan trabajar en este repositorio, también se deben descargar todas las dependencias para que se pueda hacer la conexión a la base de datos.</td></tr>
<tr><td colspan="2">Criterios de aceptación</td><td colspan="2">● Se deben configurar los permisos para que todos los integrantes tengan acceso adecuado.
● Descargar las dependencias necesarias que se necesite el proyecto</td></tr>
<tr><td colspan="4">Restricciones</td></tr>
<tr><td colspan="4">No aplica para esta historia de usuario.</td></tr>
<tr><td colspan="4"></td></tr>
</table>

<table>
<tr><th colspan="4">HISTORIA DE USUARIO</th></tr>
<tr><th colspan="4">Prioridad: ALTA</th></tr>
<tr><td>Código del Requerimiento:</td><td>NP0002</td><td>Actor:</td><td>SSP</td></tr>
<tr><td colspan="2">Nombre del Requerimiento:</td><td colspan="2">Diagrama E-R</td></tr>
<tr><th colspan="4">Descripción</th></tr>
<tr><td colspan="4">Se requiere la creación del diagrama UML para identificar las tablas necesarias en la base de datos y sus relaciones. Esto permitirá estructurar adecuadamente los datos y garantizar un diseño lógico que facilite la creación de clases en Java y su posterior implementación.</td></tr>
<tr><th colspan="4">Funcionalidad</th></tr>
<tr><td colspan="4">El diagrama UML debe estar diseñado con una adecuada normalización para garantizar la eficiencia y consistencia de los datos.
Debe incluir relaciones correctamente definidas:

● Uno a uno: Para entidades que tienen una relación exclusiva entre sí.
● Uno a muchos: Para modelar asociaciones donde una entidad está vinculada a múltiples instancias de otra.
● Muchos a muchos: Implementadas mediante tablas intermedias para asegurar un diseño óptimo.

El diagrama servirá como base para automatizar la generación de tablas en la base de datos, simplificando la creación de clases en Java y asegurando un mapeo correcto de las relaciones.</td></tr>
<tr><td>Criterios de aceptación</td><td colspan="3">● El diagrama UML debe representar correctamente todas las entidades de la base de datos con sus atributos y relaciones.
● El diagrama debe ser claro y entendible para todo el equipo de desarrollo.</td></tr>
<tr><th colspan="4">Restricciones</th></tr>
<tr><td colspan="4">El diagrama debe estar terminado y aprobado antes de iniciar la implementación de las tablas en la base de datos.</td></tr>
<tr><td colspan="4"></td></tr>
</table>

HISTORIA DE USUARIO			
Prioridad: ALTA			
Código del Requerimiento:	NP0003	**Actor:**	DLGG
Nombre del Requerimiento:		Creación de clases	
Descripción			
Se requiere la creación de las clases para que los miembros del equipo puedan colocar los atributos, métodos, getter y setter.			
Funcionalidad			
Se deben crear todas las clases para que los miembros puedan colocar el id de la clase con sus atributos para que todo sea funcional en esta clase.			
Criterios de aceptación	Crear todas las clases que se necesiten en la base de datos de la aerolínea.		
Restricciones			
No aplica para esta historia de usuario.			

HISTORIA DE USUARIO			
Prioridad: MEDIA			
Código del Requerimiento:	NP0004	**Actor:**	KSRR
Nombre del Requerimiento:		Creación de Atributos, métodos, getter y setter	
Descripción			
Se requiere la creación de los atributos en las clases, métodos, getter, setter y toString.			
Funcionalidad			
Se deben colocar los atributos que aparecen en el diagrama UML para su buen funcionamiento.			
Criterios de aceptación	• Seguir los atributos que se muestran en el diagrama.		
Restricciones			
• Los nombres de los atributos y métodos deben seguir la convención de nomenclatura **camelCase** y ser consistentes con los estándares definidos por el equipo. • Cada clase debe tener solo los atributos y métodos que sean estrictamente necesarios según los requerimientos del proyecto, evitando redundancias.			

HISTORIA DE USUARIO			
Prioridad: MEDIA			
Código del Requerimiento:	NP0005	**Actor:**	KSRR
Nombre del Requerimiento:		Creación de los repositorios	
Descripción			
Se necesita crear los repositorios de cada clase.			
Funcionalidad			
Se necesitan los repositores para que se pueda heredar los métodos de la interfaz de Jpa.			
Criterios de aceptación	• Hacer la herencia en todos los repositorios.		
Restricciones			
Todos los repositorios deben extender de Jpa			

HISTORIA DE USUARIO			
Prioridad: ALTA			
Código del Requerimiento:	NP0006	**Actor:**	DLGG
Nombre del Requerimiento:		Creación de los Servicios	
Descripción			
Se necesitan crear los servicios de todas las clases para que se pueda guardar, editar, mostrar y eliminar			
Funcionalidad			
Este servicio nos servirá para poder los datos que de las clases que hemos creado en nuestra base de datos			
Criterios de aceptación	• Hacer bien los métodos de estos servicios para que no vaya a dar error		
Restricciones			

<table>
<tr><th colspan="4">HISTORIA DE USUARIO</th></tr>
<tr><td colspan="4">Prioridad: ALTA</td></tr>
<tr><td>Código del Requerimiento:</td><td>NP0007</td><td>Actor:</td><td>SSP</td></tr>
<tr><td colspan="2">Nombre del Requerimiento:</td><td colspan="2">Creación de los controladores</td></tr>
<tr><td colspan="4">Descripción</td></tr>
<tr><td colspan="4">Se necesita crear los controladores para las clases del proyecto con el objetivo de permitir que los usuarios accedan, mediante enlaces, a la información de las tablas en formato JSON. Esto facilitará la interacción con los datos de manera estructurada y servirá como base para la comunicación con el frontend o cualquier cliente que consuma la API.</td></tr>
<tr><td colspan="4">Funcionalidad</td></tr>
<tr><td colspan="4">● Los controladores deben manejar las peticiones HTTP (GET, POST, PUT, DELETE) para interactuar con las tablas de la base de datos.
● Cada controlador debe proporcionar endpoints que devuelvan los datos de las tablas en formato JSON cuando el usuario acceda a los enlaces correspondientes.</td></tr>
<tr><td>Criterios de aceptación</td><td colspan="3">● Los endpoints deben devolver los datos de las tablas en formato JSON correctamente estructurado.
● Cada controlador debe tener al menos un método para recuperar los datos de una tabla (por ejemplo, GET para obtener todos los registros).</td></tr>
<tr><td colspan="4">Restricciones</td></tr>
<tr><td colspan="4">No se deben realizar cambios a la estructura de la base de datos sin previo acuerdo con el equipo de desarrollo y diseño.</td></tr>
</table>

HISTORIA DE USUARIO			
Prioridad: ALTA			
Código del Requerimiento:	NP0008	**Actor:**	KSRR
Nombre del Requerimiento:		Estructura hexagonal	

Descripción

Se debe reestructurar el proyecto actual para adoptar la arquitectura hexagonal (también conocida como Ports and Adapters). Esto implica la creación de tres carpetas principales:

1. **Aplicación**: Contendrá la lógica de aplicación y sus respectivos casos de uso.
2. **Dominio**: Incluirá las clases que representan el núcleo del negocio y las interfaces que definen las operaciones principales.
3. **Infraestructura**: Se encargará de las implementaciones concretas de las interfaces definidas en el dominio, como la conexión a bases de datos, servicios externos, etc.

Cada carpeta deberá incluir las clases e interfaces correspondientes para mantener la separación adecuada de responsabilidades y asegurar que el proyecto siga los principios de la arquitectura hexagonal, facilitando así su mantenimiento y escalabilidad.

Funcionalidad

Crear la estructura de carpetas Aplicación, Dominio e Infraestructura dentro del proyecto.

En la carpeta **Aplicación**, se deben incluir las clases responsables de los casos de uso o lógica de negocio de la aplicación.

Criterios de aceptación	La estructura del proyecto debe contener las tres carpetas principales: Aplicación, Dominio e Infraestructura.

Restricciones

La estructura hexagonal debe ser adoptada sin modificar las funcionalidades existentes del proyecto; solo se reestructurará el código y las capas.

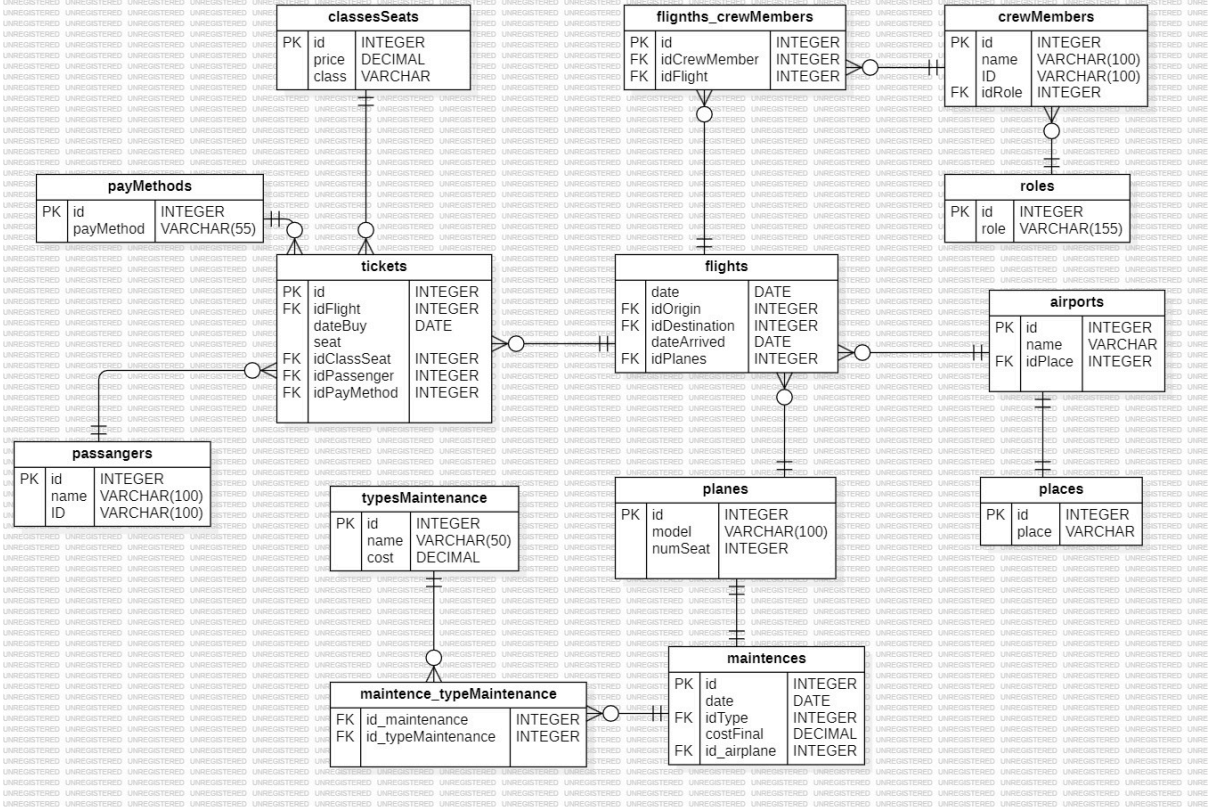
HISTORIA DE USUARIO			
Prioridad: ALTA			
Código del Requerimiento:	NP0009	**Actor:**	DLGG
Nombre del Requerimiento:		Creación de la página web	
Descripción			
Necesitamos una interfaz para que el usuario pueda interactuar con las funcionalidades que hemos creado para el API de la aerolínea			
Funcionalidad			
La página debe tener una interfaz agradable para el usuario y debe ser responsive			
Criterios de aceptación	• La página debe ser completamente responsive para que todos los usuarios puedan usar nuestra página		
Restricciones			
No usar colores que no combinen o que no sean agradables para la vista del usuario y la página debe ser completamente responsive			

HISTORIA DE USUARIO			
Prioridad: ALTA			
Código del Requerimiento:	NP0010	**Actor:**	SSP
Nombre del Requerimiento:		Creación de los endPoints	
Descripción			
Se necesita crear los endPoints para la comunicación del front-end y el back-end			
Funcionalidad			
Los endPoints son esenciales para hacer los métodos de Get, Put, Pull y Delete y para la comunicación entre la página web y Java			
Criterios de aceptación	• Se deben crear los endPoints necesarios para que se pueda hacer bien la comunicación con el front-end		
Restricciones			
Se deben crear los endPoints de todas las entidades que se tiene en la base datos			

HISTORIA DE USUARIO			
Prioridad: ALTA			
Código del Requerimiento:	NP0011	**Actor:**	DLGG
Nombre del Requerimiento:		Interacción del usuario con la página	
Descripción			
Se necesitan crear las secciones de cada entidad que tenemos para que el usuario pueda entrar y ver la información de cada entidad, pueda registrar, consultar y eliminar.			
Funcionalidad			
Esto será útil para que el usuario pueda usar nuestra API de manera mas facil, asi tambien pueda registrar algún dato, editar o eliminar.			
Criterios de aceptación	● Se debe hacer bien este enlace con los endPoints del creados en el back-end para que puedan obtener bien los datos.		
Restricciones			
El usuario no debe editar o eliminar algún dato que sea propia de la aerolínea, sólo podrá editar la información de él o editar datos que se puedan en el Ticket.			

<table>
<tr><th colspan="4">HISTORIA DE USUARIO</th></tr>
<tr><td colspan="4">Prioridad: MEDIA</td></tr>
<tr><td>Codigo del Requerimiento:</td><td>NP0012</td><td>Actor:</td><td>KSRR</td></tr>
<tr><td colspan="2">Nombre del Requerimiento:</td><td colspan="2">Creación del Readme</td></tr>
<tr><td colspan="4">Descripción</td></tr>
<tr><td colspan="4">Se debe crear un archivo README para el proyecto en el que se brinden detalles completos sobre el funcionamiento, requisitos y pasos necesarios para descargar, instalar y ejecutar el proyecto. Este archivo debe estar diseñado para que los usuarios puedan comprender rápidamente cómo utilizar el proyecto, cuáles son sus dependencias, y cómo contribuir en caso de ser necesario.</td></tr>
<tr><td colspan="4">Funcionalidad</td></tr>
<tr><td colspan="4">● El archivo README debe incluir una breve descripción del proyecto, su propósito y objetivos.
● Debe incluir los requisitos previos para ejecutar el proyecto (por ejemplo, versiones de lenguajes, herramientas, o frameworks necesarios).</td></tr>
<tr><td colspan="2">Criterios de aceptación</td><td colspan="2">● El archivo README debe estar presente en la raíz del proyecto.
● Debe incluir una descripción clara del proyecto, especificando su propósito y objetivos.
● Incluir una sección detallada de requisitos previos para ejecutar el proyecto, como herramientas o versiones necesarias.</td></tr>
<tr><td colspan="4">Restricciones</td></tr>
<tr><td colspan="4">● El README debe estar escrito en inglés, salvo que se especifique otro idioma para la documentación del proyecto.
● El archivo debe estar diseñado de forma sencilla, pero completa, evitando detalles excesivos que puedan confundir a los usuarios.</td></tr>
</table>

8. Diagrama E.R (Entidad Relación)



9.1. Vistas de Frontend (React)

El frontend en React se encargará de interactuar con el backend y mostrar la información al usuario (en este caso, los administradores). Algunas vistas clave que podrías necesitar:

1. Pantalla de inicio (Dashboard):
   En esta pantalla, los administradores podrán ver un resumen de las operaciones actuales, como el número de vuelos programados, pasajeros, y un acceso rápido a las funciones principales.
   - Componentes:
     - Resumen de vuelos (tabla o gráfica)
     - Notificaciones recientes (cancelaciones, problemas de mantenimiento, etc.)
     - Accesos rápidos a las funciones de administración (programar vuelo, gestionar reservas, etc.)
2. Gestión de Vuelos:
   Una vista donde los administradores pueden ver, modificar o cancelar vuelos.
   - Componentes:
     - Tabla de vuelos con detalles como hora, destino, avión asignado, etc.
     - Botones para agregar, editar o eliminar vuelos.
     - Filtros por fecha, estado del vuelo, etc.
     - Formularios para crear o modificar vuelos.
3. Gestión de Mantenimiento de Aeronaves:
   Vista para controlar los mantenimientos programados y realizados.
   - Componentes:
     - Tabla de mantenimientos con detalles como el avión, tipo de mantenimiento, fecha, etc.
     - Opción de programar mantenimiento o marcar como completado.
4. Gestión de Tripulación:
   Para gestionar las tripulaciones asignadas a los vuelos.
   - Componentes:
     - Lista de tripulantes y sus vuelos asignados.
     - Opción de modificar asignaciones o agregar nuevos tripulantes.

Herramientas para crear estas vistas:

- Canva para diseñar las mockups de las interfaces.
- React y Tailwind para implementar las vistas interactivas, con un diseño basado en componentes reutilizables.

10. Arquitecturas

El proyecto comenzó con la implementación de la arquitectura **Modelo-Vista-Controlador** (MVC) en Java. Este patrón de diseño divide la aplicación en tres componentes principales:

1. **Modelo**: Representa la lógica de negocio y los datos. Se encarga de acceder y manipular la base de datos o cualquier fuente de datos.
2. **Vista**: Es la interfaz de usuario (UI), que presenta los datos del modelo de manera comprensible para el usuario y le permite interactuar con ellos.
3. **Controlador**: Actúa como intermediario entre el modelo y la vista. Recibe las entradas del usuario, procesa la información (modificando el modelo si es necesario) y actualiza la vista.

Sin embargo, durante el desarrollo del proyecto, el equipo decidió cambiar la arquitectura a **Hexagonal**, utilizando la arquitectura hexagonal con **Vertical Sliders**, asegurando así un orden claro correspondiente a cada clase con su respectiva carpeta.