Contenido

Historias de Usuarios

Consideraciones Generales

Manejo de Excepciones

Es esencial implementar un sistema de manejo de excepciones que capture y gestione de forma centralizada los errores en las operaciones. Esto incluye:

- Uso de bloques try-catch y la definición de excepciones personalizadas para situaciones específicas (por ejemplo, intento de duplicación, error en la validación de datos, conflictos de horarios).
- Notificaciones automáticas a administradores o logs detallados de errores utilizando herramientas de logging (como Winston) para facilitar la identificación y resolución de incidencias sin afectar la experiencia del usuario.

Auditoría y Registro

Cada operación debe quedar registrada en un historial de auditoría. Esto implica:

- Registrar la identidad del usuario que realizó la acción, la fecha y la hora exacta, y los detalles específicos del cambio realizado.
- Utilizar un sistema de logging estructurado (por ejemplo, Winston) que se integre con herramientas de monitoreo y trazabilidad (OpenTelemetry y Sentry) para facilitar la supervisión y auditoría interna.

Tecnologías y Entorno

La solución se desarrollará utilizando tecnologías modernas y escalables, entre las cuales destacan:

- **Framework:** NestJS con TypeScript, que facilita la implementación de una arquitectura modular basada en principios de Clean Code y Arquitectura Hexagonal.
- **Base de Datos:** MongoDB gestionado mediante Prisma, permitiendo un almacenamiento flexible y consultas eficientes.
- **Integración en Monorepo:** Utilización de NX para la gestión de múltiples microservicios, facilitando la escalabilidad y mantenimiento del código.
- **Infraestructura y CI/CD:** Uso de Pulumi para la gestión de la infraestructura como código, junto con pipelines de CI/CD (GitHub Actions y SonarQube) para asegurar despliegues continuos y calidad de código.

MUST - Módulo de Gestión de Recursos

RF-01: Crear, editar y eliminar recursos.

HU-01: Crear Recurso

Historia de Usuario

Como **Administrador**, quiero **crear un recurso** para que la información de los espacios (salones, auditorios, equipos, laboratorios, etc.) esté actualizada y se puedan gestionar las reservas de manera precisa.

Criterios de Aceptación

- El formulario de creación debe permitir ingresar datos obligatorios: nombre, tipo, ubicación, capacidad, disponibilidad y, de ser necesario, reglas de uso.
- Se debe validar que:
    - Los campos obligatorios no queden vacíos.
    - La capacidad ingresada sea un número entero mayor que 0.
    - Los horarios de disponibilidad no se superpongan con períodos bloqueados.
- Una vez creada, la acción se debe registrar en un historial de auditoría (quién creó el recurso, cuándo y con qué datos).
- Se debe mostrar un mensaje de confirmación exitoso al usuario.

Tareas y Subtareas

- **Tarea 1: Diseño de Interfaz de Usuario (UI)**
    - *Subtarea 1.1:* Crear mockups y wireframes del formulario de creación de recurso.
    - *Subtarea 1.2:* Definir y validar las reglas de validación en el frontend (campos obligatorios, formato de datos).
- **Tarea 2: Desarrollo del Endpoint de Creación en NestJS**
    - *Subtarea 2.1:* Definir el DTO (Data Transfer Object) para la creación de recursos.
    - *Subtarea 2.2:* Implementar la lógica de negocio en el servicio de recursos, incluyendo validaciones (por ejemplo, que la capacidad sea mayor a 0).
    - *Subtarea 2.3:* Integrar el servicio con la base de datos usando Prisma y MongoDB.
    - *Subtarea 2.4:* Desarrollar pruebas unitarias para el endpoint.
- **Tarea 3: Implementación del Registro de Auditoría**
    - *Subtarea 3.1:* Configurar la herramienta de logging (por ejemplo, Winston) para registrar la creación del recurso.
    - *Subtarea 3.2:* Almacenar en la base de datos la información del usuario que crea el recurso y la fecha/hora de la acción.
- **Tarea 4: Pruebas de Aceptación y BDD con Jasmine**
    - *Subtarea 4.1:* Escribir escenarios Given-When-Then para la creación de un recurso.

    - *Subtarea 4.2:* Ejecutar pruebas de integración que verifiquen el flujo completo (desde la UI hasta la base de datos).
- **Tarea 5: Documentación y Despliegue**
    - *Subtarea 5.1:* Documentar el endpoint, las reglas de validación y el flujo de auditoría.
    - *Subtarea 5.2:* Integrar la funcionalidad en el pipeline de CI/CD (GitHub Actions y SonarQube).

HU-02: Editar Recurso

Historia de Usuario

Como **Administrador**, quiero **editar la información de un recurso** para mantener la base de datos actualizada y reflejar cualquier cambio en las características o disponibilidad de los espacios.

Criterios de Aceptación

- La interfaz debe mostrar un formulario de edición con los datos actuales del recurso precargados.
- Se deben poder modificar todos los atributos editables (nombre, tipo, ubicación, capacidad, horarios, etc.).
- Las validaciones deben asegurarse de que los nuevos datos cumplan los mismos criterios que en la creación (campos obligatorios, capacidad > 0, etc.).
- Toda modificación debe registrarse en el historial de auditoría (incluyendo qué cambios se realizaron y por quién).
- Al finalizar la edición, se debe notificar al usuario con un mensaje de éxito.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Edición**
    - *Subtarea 1.1:* Diseñar y aprobar mockups de la pantalla de edición de recurso.
    - *Subtarea 1.2:* Asegurarse de que el formulario pre-llene los datos actuales y permita cambios de manera intuitiva.
- **Tarea 2: Desarrollo del Endpoint de Edición en NestJS**
    - *Subtarea 2.1:* Definir el DTO para la actualización de recursos.
    - *Subtarea 2.2:* Implementar la lógica de actualización en el servicio, aplicando validaciones pertinentes.
    - *Subtarea 2.3:* Integrar la actualización con la base de datos.
    - *Subtarea 2.4:* Desarrollar pruebas unitarias para este flujo.
- **Tarea 3: Registro de Cambios en el Historial de Auditoría**
    - *Subtarea 3.1:* Implementar la funcionalidad para registrar los cambios (quién, cuándo, qué se cambió).
    - *Subtarea 3.2:* Asegurar la trazabilidad de todas las modificaciones.
- **Tarea 4: Pruebas de Aceptación y BDD con Jasmine**
    - *Subtarea 4.1:* Escribir escenarios BDD (Given-When-Then) que cubran el flujo de edición.

    - *Subtarea 4.2:* Ejecutar pruebas de integración para verificar la actualización en la base de datos.
- **Tarea 5: Documentación y Despliegue**
    - *Subtarea 5.1:* Actualizar la documentación técnica y del usuario.
    - *Subtarea 5.2:* Incluir este flujo en el pipeline de CI/CD.

HU-03: Eliminar o Deshabilitar Recurso

Historia de Usuario

Como **Administrador**, quiero **eliminar o deshabilitar un recurso** para evitar que se utilicen espacios obsoletos o que ya no deben estar disponibles, asegurando la integridad de las reservas existentes.

Criterios de Aceptación

- Si el recurso no tiene reservas activas, el sistema debe permitir su eliminación completa.
- Si el recurso tiene reservas activas o historial de reservas, el sistema debe impedir la eliminación y, en su lugar, ofrecer la opción de deshabilitarlo.
- Al realizar la acción (eliminación o deshabilitación), se debe mostrar una confirmación y registrar la acción en el historial de auditoría.
- Se debe validar que no se puedan eliminar recursos críticos sin previo aviso a los usuarios afectados (por ejemplo, mediante una confirmación modal).

Tareas y Subtareas

- **Tarea 1: Diseño del Flujo de Eliminación/Deshabilitación**
    - *Subtarea 1.1:* Diseñar la interfaz de usuario que permita seleccionar un recurso para eliminación/deshabilitación.
    - *Subtarea 1.2:* Crear un modal de confirmación que indique los riesgos y permita elegir entre eliminar o deshabilitar.
- **Tarea 2: Desarrollo del Endpoint de Eliminación/Deshabilitación en NestJS**
    - *Subtarea 2.1:* Implementar la lógica para verificar si el recurso tiene reservas activas.
    - *Subtarea 2.2:* Si no existen reservas, proceder con la eliminación; si existen, cambiar el estado del recurso a “deshabilitado”.
    - *Subtarea 2.3:* Integrar la lógica con la base de datos y actualizar el registro del recurso.
    - *Subtarea 2.4:* Desarrollar pruebas unitarias para validar ambos flujos (eliminación y deshabilitación).
- **Tarea 3: Registro de Auditoría para la Acción**
    - *Subtarea 3.1:* Configurar el registro de logs para capturar la acción de eliminación o deshabilitación, incluyendo el usuario que realiza la acción y la fecha/hora.
- **Tarea 4: Pruebas de Aceptación y BDD con Jasmine**
    - *Subtarea 4.1:* Definir escenarios BDD (Given-When-Then) para la eliminación y para el flujo de deshabilitación.

    - *Subtarea 4.2:* Ejecutar pruebas de integración que verifiquen la integridad de la información y la actualización del historial.
- **Tarea 5: Documentación y Despliegue**
    - *Subtarea 5.1:* Documentar el comportamiento del sistema en ambos casos y las condiciones que determinan cada flujo.
    - *Subtarea 5.2:* Incluir esta funcionalidad en el pipeline de CI/CD y actualizar la documentación de usuario.

RF-03: Definir atributos clave (nombre, descripción, ubicación, capacidad, horarios de disponibilidad y reglas de uso) para garantizar la integridad de la información

HU-04: Definir Atributos Clave del Recurso

Historia de Usuario:

Como **Administrador**, quiero **definir los atributos clave de cada recurso (nombre, descripción, ubicación, capacidad, horarios de disponibilidad y reglas de uso)** para que la información de los recursos sea completa, consistente y facilite la gestión y reserva de espacios.

Criterios de Aceptación

- Se deben poder ingresar y editar los atributos básicos: nombre, descripción, ubicación y capacidad.
- La capacidad debe validarse para que sea un número entero mayor a 0.
- Se debe permitir configurar rangos de horarios de disponibilidad sin solapamientos y con validación de periodos bloqueados.
- Se deben definir reglas de uso (ej.: tiempo mínimo/máximo de reserva, restricciones y requisitos previos) que se puedan configurar y validar.
- Todos los cambios deben registrarse en el historial de auditoría, indicando quién, cuándo y qué modificaciones se realizaron.
- La información de los atributos configurados debe mostrarse en la vista de detalles del recurso.

Para abordar este requerimiento, se desglosa en tres sub-historias:

SubHU-04.1: Definir Atributos Básicos

Historia de Usuario

Como **Administrador**, quiero **definir los atributos básicos (nombre, descripción, ubicación y capacidad)** de cada recurso para que éste se identifique de manera clara y se garantice la integridad de la información.

Criterios de Aceptación

- Los campos de nombre, descripción, ubicación y capacidad son obligatorios.

- La capacidad debe validarse para que sea un número entero mayor que 0.
- Los datos ingresados deben visualizarse correctamente en la consulta de recursos.
- Toda creación o modificación de estos atributos debe registrarse en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Atributos Básicos**
  - Subtarea 1.1: Crear wireframes/mockups del formulario para ingresar y editar atributos básicos.
  - Subtarea 1.2: Validar el diseño con stakeholders y obtener feedback.
- **Tarea 2: Desarrollo del Endpoint de Atributos Básicos en NestJS**
  - Subtarea 2.1: Definir el DTO (Data Transfer Object) para la creación/actualización de atributos básicos.
  - Subtarea 2.2: Implementar la lógica en el servicio que gestione la creación y actualización, incluyendo validaciones (por ejemplo, capacidad > 0).
  - Subtarea 2.3: Integrar el endpoint con la base de datos usando Prisma y MongoDB.
- **Tarea 3: Implementación de Validaciones y Registro de Auditoría**
  - Subtarea 3.1: Desarrollar validaciones en el backend y, de ser posible, en el frontend para garantizar que los campos obligatorios estén completos.
  - Subtarea 3.2: Configurar el registro de auditoría (usando Winston u otra herramienta) para almacenar información sobre quién creó o editó el recurso y cuándo.
- **Tarea 4: Pruebas y Documentación**
  - Subtarea 4.1: Escribir pruebas unitarias y de integración en Jasmine, aplicando escenarios Given-When-Then (BDD) para el flujo de creación/edición.
  - Subtarea 4.2: Documentar la funcionalidad y actualizar la guía del usuario.

SubHU-04.2: Configurar Horarios de Disponibilidad

Historia de Usuario

Como **Administrador**, quiero **configurar los horarios de disponibilidad de cada recurso** para asegurar que solo se puedan reservar en períodos válidos y evitar conflictos de agenda.

Criterios de Aceptación

- Se debe permitir definir rangos horarios de disponibilidad para cada recurso.
- No deben permitirse solapamientos entre los horarios ingresados ni conflictos con períodos bloqueados.
- La interfaz debe mostrar una vista clara (por ejemplo, un calendario o lista) de los horarios configurados.
- Los cambios en la configuración de horarios deben registrarse en el historial de auditoría.
- (Opcional) Integración con un sistema de calendario para validar disponibilidad en tiempo real.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Horarios**
    - Subtarea 1.1: Crear mockups de la pantalla para la configuración de horarios de disponibilidad.
    - Subtarea 1.2: Recoger y validar requerimientos de visualización con stakeholders.
- **Tarea 2: Desarrollo del Módulo de Gestión de Horarios en NestJS**
    - Subtarea 2.1: Definir el DTO para la gestión de horarios (incluyendo fecha, hora de inicio y fin).
    - Subtarea 2.2: Implementar la lógica en el servicio para agregar, editar y eliminar rangos de horarios.
    - Subtarea 2.3: Implementar validaciones para evitar solapamientos y garantizar el cumplimiento de periodos bloqueados.
- **Tarea 3: Integración y Validación en Tiempo Real**
    - Subtarea 3.1: (Opcional) Integrar con APIs de calendario para sincronizar reservas y validar disponibilidad.
    - Subtarea 3.2: Desarrollar pruebas de integración que verifiquen la correcta validación de horarios.
- **Tarea 4: Pruebas y Documentación**
    - Subtarea 4.1: Escribir pruebas unitarias y escenarios BDD en Jasmine.
    - Subtarea 4.2: Documentar la configuración de horarios y actualizar la documentación técnica.

SubHU-04.3: Establecer Reglas de Uso

Historia de Usuario

Como **Administrador**, quiero **establecer reglas de uso para cada recurso** para regular su utilización, asegurar el cumplimiento de políticas institucionales y evitar abusos en las reservas.

Criterios de Aceptación

- Se debe permitir definir reglas de uso, tales como tiempo mínimo y máximo de reserva, restricciones de uso y requisitos previos.
- La interfaz debe ser intuitiva para configurar y editar estas reglas.
- Las reglas definidas deben validarse para evitar conflictos con la disponibilidad y otros atributos del recurso.
- Toda modificación en las reglas debe registrarse en el historial de auditoría.
- La información de las reglas de uso debe mostrarse en la vista de detalles del recurso.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz para Reglas de Uso**
    - Subtarea 1.1: Crear wireframes/mockups para la configuración de reglas de uso.

    - Subtarea 1.2: Validar el diseño con usuarios clave (por ejemplo, administradores y responsables de reservas).
- **Tarea 2: Desarrollo del Endpoint para Reglas de Uso en NestJS**
    - Subtarea 2.1: Definir el DTO y el modelo de datos para las reglas de uso.
    - Subtarea 2.2: Implementar la lógica de negocio en el servicio para agregar, editar y eliminar reglas.
    - Subtarea 2.3: Integrar esta funcionalidad con la base de datos.
- **Tarea 3: Validación de Reglas y Manejo de Excepciones**
    - Subtarea 3.1: Desarrollar validaciones para asegurar que las reglas configuradas no entren en conflicto con otros parámetros (por ejemplo, horarios o capacidad).
    - Subtarea 3.2: Implementar manejo de errores y excepciones que notifiquen al usuario en caso de conflictos.
- **Tarea 4: Pruebas y Documentación**
    - Subtarea 4.1: Desarrollar pruebas unitarias y escenarios BDD en Jasmine para el módulo de reglas de uso.
    - Subtarea 4.2: Documentar la funcionalidad, incluyendo ejemplos de configuración de reglas y procedimientos de validación.

RF-05: Configuración de reglas de disponibilidad esenciales para evitar conflictos en la asignación de recursos

HU-05: Configuración de Reglas de Disponibilidad

Historia de Usuario

Como **Administrador**, quiero **configurar las reglas de disponibilidad para los recursos** para garantizar que las reservas se ajusten a las normativas institucionales y optimicen el uso de los espacios, evitando conflictos y garantizando períodos de preparación adecuados.

Criterios de Aceptación Generales

- El sistema debe permitir definir, editar y eliminar reglas que establezcan:
    - Tiempo mínimo y máximo de reserva.
    - Intervalos de tiempo de preparación entre reservas.
    - Períodos bloqueados (por mantenimiento, eventos u otras restricciones).
    - Reglas de prioridad de uso según el perfil del usuario.
- Todas las configuraciones deben contar con validaciones (por ejemplo, no permitir tiempos negativos o superposición de períodos).
- Cualquier cambio debe quedar registrado en el historial de auditoría (usuario, fecha, cambios realizados).
- La interfaz debe mostrar de forma clara las reglas configuradas y permitir su consulta y edición.
- Las reglas deben integrarse con el proceso de validación de reservas en tiempo real.

Para cubrir la complejidad del RF-05, se desglosa en las siguientes sub-historias:

SubHU-05.1: Configurar Tiempos de Reserva

Historia de Usuario

Como **Administrador**, quiero **definir los tiempos mínimos y máximos de reserva, así como los intervalos de preparación entre reservas** para asegurar que cada espacio tenga el tiempo suficiente para ser preparado y evitar reservas de corta duración que puedan afectar la operatividad.

Criterios de Aceptación

- Se debe permitir ingresar un tiempo mínimo de reserva (por ejemplo, 30 minutos) y un tiempo máximo (por ejemplo, 4 horas).
- Se debe poder configurar un intervalo obligatorio entre reservas (por ejemplo, 15 minutos) para tareas de preparación.
- Los valores ingresados deben validarse:
    - El tiempo mínimo debe ser menor que el máximo.
    - Los intervalos deben ser números positivos.
- La configuración se mostrará en la vista de detalles del recurso y cualquier modificación se registrará en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz para Tiempos de Reserva**
    - *Subtarea 1.1:* Crear wireframes y mockups del formulario de configuración de tiempos (mínimo, máximo y preparación).
    - *Subtarea 1.2:* Validar el diseño con stakeholders y ajustar según feedback.
- **Tarea 2: Desarrollo del Endpoint en NestJS**
    - *Subtarea 2.1:* Definir el DTO para la configuración de tiempos de reserva.
    - *Subtarea 2.2:* Implementar la lógica en el servicio de recursos para guardar y actualizar estos valores.
    - *Subtarea 2.3:* Realizar las validaciones: verificar que el tiempo mínimo < tiempo máximo y que los intervalos sean válidos.
    - *Subtarea 2.4:* Integrar con la base de datos utilizando Prisma y MongoDB.
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
    - *Subtarea 3.1:* Configurar el registro de cambios (quién, cuándo, qué se modificó) usando un sistema de logging (p. ej., Winston).
    - *Subtarea 3.2:* Implementar manejo de errores para notificar al usuario si se ingresan datos inválidos.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Escribir pruebas unitarias e integración (usando Jasmine y escenarios BDD Given-When-Then) para el flujo de configuración.
    - *Subtarea 4.2:* Documentar la funcionalidad y actualizar la guía del usuario.

SubHU-05.2: Configurar Períodos Bloqueados

Historia de Usuario

Como **Administrador**, quiero **configurar períodos bloqueados para cada recurso** para evitar reservas durante tiempos no operativos (por ejemplo, mantenimiento, eventos institucionales o días festivos) y garantizar la integridad de la agenda.

Criterios de Aceptación

- Se debe permitir ingresar uno o más períodos bloqueados con fecha, hora de inicio y fin.
- No se deben permitir superposiciones entre períodos bloqueados o con horarios de reserva previamente definidos.
- La interfaz debe mostrar claramente los períodos bloqueados en un calendario o lista.
- Los cambios en los períodos bloqueados deben registrarse en el historial de auditoría.
- En el proceso de reserva, el sistema debe rechazar automáticamente reservas que coincidan con estos períodos.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz para Períodos Bloqueados**
  - *Subtarea 1.1:* Crear prototipos de la pantalla donde se visualizarán y configurarán los períodos bloqueados.
  - *Subtarea 1.2:* Validar el diseño con el equipo de administración y usuarios clave.
- **Tarea 2: Desarrollo del Módulo de Períodos Bloqueados en NestJS**
  - *Subtarea 2.1:* Definir el DTO y modelo de datos para un período bloqueado.
  - *Subtarea 2.2:* Implementar la lógica para agregar, editar y eliminar períodos bloqueados.
  - *Subtarea 2.3:* Desarrollar validaciones que eviten la superposición de períodos bloqueados y su conflicto con horarios de reserva.
- **Tarea 3: Integración y Pruebas de Validación en Reserva**
  - *Subtarea 3.1:* Integrar el módulo de períodos bloqueados con el servicio de validación de reservas.
  - *Subtarea 3.2:* Escribir pruebas unitarias y de integración (usando Jasmine) que simulen reservas en períodos bloqueados y verifiquen el rechazo correcto.
- **Tarea 4: Documentación y Registro de Auditoría**
  - *Subtarea 4.1:* Documentar la funcionalidad y su integración en el sistema.
  - *Subtarea 4.2:* Configurar el registro de auditoría para cada acción relacionada con la gestión de períodos bloqueados.

SubHU-05.3: Configurar Prioridades de Uso

Historia de Usuario

Como **Administrador**, quiero **configurar reglas de prioridad de uso para los recursos** para asignar preferencia en las reservas a ciertos perfiles (por ejemplo, docentes o administrativos) y gestionar la asignación en situaciones de alta demanda.

Criterios de Aceptación

- Se debe permitir definir niveles de prioridad (ej.: alta, media, baja) para diferentes tipos de usuarios o solicitudes.

- La interfaz debe ofrecer una opción para asignar prioridades de uso al configurar o editar un recurso.
- Durante la validación de una reserva, el sistema debe tener en cuenta la prioridad configurada para sugerir alternativas o gestionar listas de espera.
- Las configuraciones de prioridad deben poder modificarse y quedar registradas en el historial de auditoría.
- La configuración debe integrarse con el mecanismo de notificación, informando a los usuarios de cualquier ajuste de prioridad.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz para Prioridades de Uso**
    - *Subtarea 1.1:* Diseñar mockups del formulario de asignación de prioridad al recurso.
    - *Subtarea 1.2:* Recoger feedback de los stakeholders sobre los niveles de prioridad requeridos.
- **Tarea 2: Desarrollo del Endpoint para Configuración de Prioridades en NestJS**
    - *Subtarea 2.1:* Definir el DTO y modelo de datos para las reglas de prioridad.
    - *Subtarea 2.2:* Implementar la lógica en el servicio para asignar, editar y eliminar prioridades.
    - *Subtarea 2.3:* Integrar la funcionalidad con el sistema de reservas para que se valide la prioridad durante la asignación.
- **Tarea 3: Integración y Pruebas de Validación de Prioridades**
    - *Subtarea 3.1:* Desarrollar pruebas unitarias y de integración (usando Jasmine y el patrón BDD) para verificar el correcto funcionamiento de las reglas de prioridad en diferentes escenarios de reserva.
    - *Subtarea 3.2:* Validar que los cambios en las prioridades se reflejen correctamente en la lógica de asignación de reservas.
- **Tarea 4: Documentación y Registro de Auditoría**
    - *Subtarea 4.1:* Documentar la configuración de prioridades, incluyendo ejemplos de asignación.
    - *Subtarea 4.2:* Asegurar que cada modificación en las reglas de prioridad quede registrada en el historial de auditoría.

SHOULD - Módulo de Gestión de Recursos

RF-02: Asociar cada recurso a una categoría y a uno o más programas académicos, lo que permite clasificar y filtrar la información de manera efectiva

HU-06: Asociar Recurso a Categoría y Programa Académico

Historia de Usuario

Como **Administrador**, quiero **asociar cada recurso a una categoría y a uno o más programas académicos** para que la información se organice adecuadamente, facilitando la búsqueda, filtrado y reserva según las necesidades académicas y administrativas.

Criterios de Aceptación

- El sistema debe disponer de una lista de categorías predefinidas (por ejemplo, Salón, Laboratorio, Auditorio, Equipo, etc.) que pueda ser configurada.
- Se debe permitir asignar a cada recurso uno o varios programas académicos.
- En el formulario de creación y edición de recursos, el usuario debe poder seleccionar al menos una categoría y un programa académico.
- Si se intenta guardar un recurso sin al menos una categoría y un programa asignados, el sistema mostrará un error claro.
- Cualquier cambio en la asociación (asignación o actualización) deberá registrarse en el historial de auditoría, indicando quién realizó el cambio y en qué momento.
- La vista de detalles del recurso debe mostrar claramente la categoría y los programas académicos asociados.

Para facilitar la implementación y validación, se desglosa en dos sub-historias:

SubHU-06.1: Asignación de Categoría al Recurso

Historia de Usuario

Como **Administrador**, quiero **asignar una categoría a cada recurso** para clasificar y organizar los espacios de manera coherente y facilitar su búsqueda y filtrado.

Criterios de Aceptación

- Debe existir un listado de categorías configurables que se muestren en un componente select (desplegable o tags).
- El sistema obligará a seleccionar al menos una categoría al crear o editar un recurso.
- La categoría asignada se mostrará en la vista de detalles del recurso.
- Cualquier modificación en la categoría deberá registrarse en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Selección de Categoría**
    - Subtarea 1.1: Crear wireframes y mockups del formulario de recurso que incluya la selección de categoría.
    - Subtarea 1.2: Validar el diseño con stakeholders y ajustar según feedback.
- **Tarea 2: Desarrollo del Endpoint para Asignación de Categoría**
    - Subtarea 2.1: Definir el DTO para la asociación de categoría.
    - Subtarea 2.2: Implementar la lógica en el servicio de recursos para guardar la categoría seleccionada.
    - Subtarea 2.3: Validar en el backend que al menos se seleccione una categoría.
    - Subtarea 2.4: Desarrollar pruebas unitarias para el endpoint.
- **Tarea 3: Integración con Base de Datos y Registro de Auditoría**
    - Subtarea 3.1: Actualizar el modelo de datos para incluir la relación entre recursos y categorías.
    - Subtarea 3.2: Realizar la migración usando Prisma.
    - Subtarea 3.3: Configurar el logging para registrar la acción (usuario, fecha, categoría asignada).
- **Tarea 4: Pruebas de Aceptación y Documentación**
    - Subtarea 4.1: Escribir escenarios BDD (Given-When-Then) en Jasmine para la asignación de categoría.
    - Subtarea 4.2: Documentar la funcionalidad en la guía del usuario y la documentación técnica.
    - Subtarea 4.3: Integrar el flujo en el pipeline de CI/CD.

SubHU-06.2: Asignación de Programa Académico al Recurso

Historia de Usuario

Como **Administrador**, quiero **asignar uno o más programas académicos a cada recurso** para que se pueda filtrar y gestionar el acceso a los espacios según las necesidades institucionales.

Criterios de Aceptación

- El formulario de creación/edición debe incluir un componente que permita la selección múltiple de programas académicos.
- Se debe validar que al menos se asigne un programa académico al recurso.
- La información de los programas asignados se mostrará en la vista de detalles del recurso.
- Las modificaciones en la asociación de programas deben registrarse en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz para Selección de Programas Académicos**
    - Subtarea 1.1: Crear mockups y wireframes del componente de selección múltiple de programas académicos.

    - Subtarea 1.2: Validar el diseño con stakeholders y ajustar según requerimientos.
- **Tarea 2: Desarrollo del Endpoint para Asociación de Programas Académicos**
    - Subtarea 2.1: Definir el DTO para la asociación de programas académicos.
    - Subtarea 2.2: Implementar la lógica en el servicio de recursos para almacenar la asociación.
    - Subtarea 2.3: Incluir validaciones que obliguen a seleccionar al menos un programa.
    - Subtarea 2.4: Crear pruebas unitarias que verifiquen la correcta asociación.
- **Tarea 3: Integración y Actualización de Modelo de Datos**
    - Subtarea 3.1: Modificar el modelo de datos para representar la relación de uno a muchos (o muchos a muchos) entre recursos y programas académicos.
    - Subtarea 3.2: Ejecutar migraciones con Prisma.
    - Subtarea 3.3: Implementar el registro de auditoría para cambios en la asociación.
- **Tarea 4: Pruebas de Aceptación y Documentación**
    - Subtarea 4.1: Desarrollar escenarios BDD (Given-When-Then) para el flujo de asociación de programas.
    - Subtarea 4.2: Documentar el proceso y actualizar las guías de usuario y técnica.
    - Subtarea 4.3: Integrar la funcionalidad en el pipeline de CI/CD.

RF-04: Importación masiva de recursos mediante CSV o integración con sistemas universitarios, para agilizar la carga inicial de datos

HU-07: Importación Masiva de Recursos

Historia de Usuario

Como **Administrador**, quiero **importar masivamente recursos mediante archivos CSV o integrarlos desde sistemas universitarios existentes** para agilizar la carga inicial de datos y mantener la información actualizada sin intervención manual constante.

Criterios de Aceptación Generales

- El sistema debe permitir importar un archivo CSV con una estructura predefinida que incluya todos los campos obligatorios.
- El sistema debe validar la integridad y formato de los datos del archivo CSV, mostrando errores o advertencias en caso de inconsistencias.
- Debe mostrarse un resumen final con el número de registros importados, rechazados y detalles de errores.
- En caso de errores masivos, debe ofrecerse la opción de deshacer la importación.
- El sistema debe permitir la configuración de integración con sistemas universitarios externos (por ejemplo, mediante APIs o conexión a bases de datos) para importar datos de forma programada.
- Todas las acciones de importación (CSV o integración) deberán quedar registradas en el historial de auditoría (usuario, fecha, acciones realizadas).

Dado lo anterior, se desglosa en dos sub-historias:

SubHU-07.1: Importación de Recursos desde Archivos CSV

Historia de Usuario

Como **Administrador**, quiero **importar recursos mediante archivos CSV** para cargar en bloque datos de forma rápida y precisa.

Criterios de Aceptación

- Se debe permitir seleccionar y subir un archivo CSV a través de una interfaz intuitiva.
- El sistema debe validar que el archivo CSV contenga todos los campos obligatorios (por ejemplo, nombre, tipo, ubicación, capacidad, etc.).
- Debe mostrarse una vista previa de los datos y los posibles errores (campos faltantes, formato incorrecto, etc.) antes de confirmar la importación.
- Al confirmar, se debe generar un resumen con el total de registros procesados, importados correctamente y los rechazados.
- Si se detectan errores críticos, el sistema debe permitir cancelar o revertir la importación.
- La acción de importación (éxito o errores) debe registrarse en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Importación CSV**
  - *Subtarea 1.1:* Crear wireframes y mockups del módulo de importación de CSV, incluyendo la opción de vista previa.
  - *Subtarea 1.2:* Validar el diseño con stakeholders (administradores y usuarios clave).
- **Tarea 2: Desarrollo del Módulo de Parseo y Validación de CSV**
  - *Subtarea 2.1:* Definir el formato y la estructura del archivo CSV (especificar columnas obligatorias y opcionales).
  - *Subtarea 2.2:* Implementar un parser en NestJS que lea y transforme los datos del CSV en objetos de datos.
  - *Subtarea 2.3:* Desarrollar validaciones en el backend para comprobar la integridad y formato de cada registro (por ejemplo, validación de capacidad numérica, campos no vacíos, etc.).
  - *Subtarea 2.4:* Implementar manejo de errores que capture y reporte registros con inconsistencias.
- **Tarea 3: Generación de Resumen y Gestión de Reversiones**
  - *Subtarea 3.1:* Crear lógica para generar un resumen final con el número de registros importados y rechazados, indicando los errores encontrados.
  - *Subtarea 3.2:* Desarrollar la opción de deshacer la importación en caso de errores masivos, permitiendo la corrección y reimportación.
- **Tarea 4: Registro de Auditoría y Manejo de Excepciones**
  - *Subtarea 4.1:* Configurar el logging (por ejemplo, utilizando Winston) para registrar cada acción de importación, incluyendo usuario, fecha y resultados.

    - *Subtarea 4.2:* Implementar manejo de excepciones que notifique al usuario ante fallos críticos en el proceso.
- **Tarea 5: Pruebas y Documentación**
    - *Subtarea 5.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine con escenarios Given-When-Then (BDD) para validar el flujo de importación CSV.
    - *Subtarea 5.2:* Documentar la funcionalidad, describiendo el formato del CSV, instrucciones de uso y ejemplos de error.
    - *Subtarea 5.3:* Incluir este módulo en el pipeline de CI/CD (GitHub Actions, SonarQube, etc.).

SubHU-07.2: Integración con Sistemas Universitarios Existentes

Historia de Usuario

Como **Administrador**, quiero **integrar el sistema con los sistemas universitarios existentes** para importar y sincronizar automáticamente los recursos, asegurando que la información se mantenga actualizada sin intervención manual.

Criterios de Aceptación

- Se debe disponer de una interfaz de configuración para establecer la conexión con sistemas externos (por ejemplo, APIs, bases de datos) mediante credenciales seguras.
- El sistema debe permitir definir la frecuencia de sincronización (diaria, semanal, mensual).
- Durante la sincronización, se deben validar y mapear correctamente los datos importados al modelo de recursos.
- En caso de datos duplicados o inconsistentes, el sistema debe notificar al administrador y registrar los incidentes.
- Debe generarse un log/resumen de cada sincronización, incluyendo la cantidad de registros actualizados, nuevos o rechazados.
- La acción de sincronización debe quedar registrada en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Configuración de Integración**
    - *Subtarea 1.1:* Crear wireframes/mockups para la pantalla de configuración de la conexión con sistemas universitarios (campos de API, credenciales, frecuencia).
    - *Subtarea 1.2:* Validar el diseño con stakeholders y ajustar según requerimientos.
- **Tarea 2: Desarrollo de Conectores/Adaptadores**
    - *Subtarea 2.1:* Definir el modelo de datos y el mapeo de los campos provenientes del sistema externo hacia el modelo de recursos.
    - *Subtarea 2.2:* Implementar un conector en NestJS para consumir la API o conectarse a la base de datos externa.
    - *Subtarea 2.3:* Desarrollar la lógica de sincronización programada, permitiendo configurar la frecuencia de actualización.

- **Tarea 3: Validación y Manejo de Datos Importados**
    - *Subtarea 3.1:* Implementar validaciones para comprobar la integridad de los datos importados (por ejemplo, que se cumplan los formatos y campos obligatorios).
    - *Subtarea 3.2:* Desarrollar la lógica para detectar y evitar duplicados o inconsistencias durante la sincronización.
    - *Subtarea 3.3:* Configurar manejo de errores y notificaciones en caso de fallos en la integración.
- **Tarea 4: Registro de Auditoría y Resumen de Sincronización**
    - *Subtarea 4.1:* Configurar el registro de auditoría para las acciones de sincronización, registrando quién, cuándo y qué datos se actualizaron.
    - *Subtarea 4.2:* Desarrollar la generación de un resumen de sincronización que se muestre en la interfaz de configuración y se envíe por correo al administrador (opcional).
- **Tarea 5: Pruebas y Documentación**
    - *Subtarea 5.1:* Desarrollar pruebas unitarias e integración (BDD con Jasmine) para validar el proceso de sincronización.
    - *Subtarea 5.2:* Documentar la integración, especificando la configuración, el formato esperado de los datos externos y ejemplos de resolución de errores.
    - *Subtarea 5.3:* Incluir la funcionalidad en el pipeline de CI/CD para pruebas continuas y despliegue.

COULD - Módulo de Gestión de Recursos

RF-06: Módulo de mantenimiento de recursos (registro de daños, mantenimientos programados y reportes de incidentes), que aporta valor adicional pero puede implementarse en una fase posterior.

HU-08: Gestión de Mantenimiento de Recursos

Historia de Usuario

Como **Administrador**, quiero **gestionar el mantenimiento de los recursos** para garantizar su disponibilidad y correcto funcionamiento, minimizando el impacto de fallas en la operación académica.

Criterios de Aceptación Generales

- El sistema debe permitir registrar, visualizar y gestionar mantenimientos preventivos y correctivos de los recursos.
- Se debe registrar el estado del recurso antes y después del mantenimiento.
- Debe existir una integración con la disponibilidad de reservas para bloquear el uso de un recurso en mantenimiento.
- Todos los mantenimientos registrados deben quedar almacenados con trazabilidad y auditoría.
- Se debe notificar a los usuarios afectados en caso de que un mantenimiento interfiera con sus reservas.

Dado lo anterior, se desglosa en las siguientes sub-historias:

SubHU-08.1: Registro de Daños e Incidentes

Historia de Usuario

Como **Usuario o Administrador**, quiero **reportar daños o incidentes en los recursos** para que se puedan programar mantenimientos y evitar su uso en condiciones inadecuadas.

Criterios de Aceptación

- Los usuarios deben poder registrar un daño o incidente asociado a un recurso.
- El formulario de reporte debe incluir:
    - Descripción del problema.
    - Nivel de criticidad (baja, media, alta).
    - Evidencias (opcional, imágenes/documentos).
- El sistema debe permitir a los administradores visualizar y gestionar los reportes.
- En caso de daños graves, el sistema debe deshabilitar automáticamente el recurso y notificar al equipo de mantenimiento.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Reporte de Daños**
    - *Subtarea 1.1:* Crear wireframes y mockups del formulario de reporte.
    - *Subtarea 1.2:* Validar la interfaz con stakeholders.
- **Tarea 2: Desarrollo del Módulo de Reporte en NestJS**
    - *Subtarea 2.1:* Definir el DTO y modelo de datos para reportes de daños.
    - *Subtarea 2.2:* Implementar API para el registro de reportes.
    - *Subtarea 2.3:* Integrar validaciones en el backend (por ejemplo, nivel de criticidad requerido).
    - *Subtarea 2.4:* Almacenar reportes en MongoDB usando Prisma.
- **Tarea 3: Registro de Auditoría y Notificaciones**
    - *Subtarea 3.1:* Implementar el registro de auditoría para cada reporte.
    - *Subtarea 3.2:* Configurar notificaciones automáticas para el equipo de mantenimiento cuando un reporte sea crítico.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Escribir pruebas unitarias e integración con Jasmine y escenarios Given-When-Then.
    - *Subtarea 4.2:* Documentar el proceso de reporte de daños.
    - *Subtarea 4.3:* Incluir el módulo en CI/CD.

SubHU-08.2: Programación de Mantenimientos Preventivos

Historia de Usuario

Como **Administrador o Técnico de Mantenimiento**, quiero **programar mantenimientos preventivos en los recursos** para minimizar la probabilidad de fallas y garantizar su óptimo funcionamiento.

Criterios de Aceptación

- El sistema debe permitir programar mantenimientos preventivos de forma periódica o específica.
- Se deben definir parámetros como:
    - Fecha y hora del mantenimiento.
    - Tipo de mantenimiento (limpieza, revisión técnica, actualización de software, etc.).
    - Técnico asignado.
- Se debe bloquear la disponibilidad del recurso durante el período de mantenimiento.
- Se debe notificar a los usuarios con reservas afectadas.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Programación de Mantenimientos**
    - *Subtarea 1.1:* Crear wireframes y mockups del módulo de mantenimiento preventivo.
    - *Subtarea 1.2:* Validar el diseño con el equipo de mantenimiento.
- **Tarea 2: Desarrollo del Módulo de Programación en NestJS**

    - *Subtarea 2.1:* Definir DTO y modelo de datos para los mantenimientos preventivos.
    - *Subtarea 2.2:* Implementar API para crear, actualizar y eliminar mantenimientos preventivos.
    - *Subtarea 2.3:* Integrar con la disponibilidad del recurso para bloquear reservas.
- **Tarea 3: Notificación y Registro de Auditoría**
    - *Subtarea 3.1:* Implementar notificaciones automáticas para los usuarios afectados.
    - *Subtarea 3.2:* Registrar en la auditoría cada acción sobre mantenimientos programados.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias y de integración con Jasmine.
    - *Subtarea 4.2:* Documentar la configuración de mantenimientos preventivos.

SubHU-08.3: Registro y Seguimiento de Mantenimientos Correctivos

Historia de Usuario

Como **Administrador o Técnico de Mantenimiento**, quiero **registrar y dar seguimiento a los mantenimientos correctivos** para asegurar que los recursos dañados sean reparados y vuelvan a estar disponibles.

Criterios de Aceptación

- Se debe permitir registrar mantenimientos correctivos indicando:
    - Recurso afectado.
    - Motivo del mantenimiento.
    - Acciones realizadas.
    - Fecha y duración del mantenimiento.
    - Técnico responsable.
- Los mantenimientos correctivos deben actualizar el estado del recurso (por ejemplo, de “en mantenimiento” a “operativo”).
- Los administradores deben poder consultar el historial de mantenimientos correctivos de cada recurso.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Registro de Mantenimientos Correctivos**
    - *Subtarea 1.1:* Crear mockups para el registro y visualización de mantenimientos correctivos.
    - *Subtarea 1.2:* Validar con el equipo de mantenimiento.
- **Tarea 2: Desarrollo del Módulo de Mantenimientos Correctivos en NestJS**
    - *Subtarea 2.1:* Definir DTO y modelo de datos para registros de mantenimiento.
    - *Subtarea 2.2:* Implementar API para registrar, actualizar y consultar mantenimientos correctivos.
    - *Subtarea 2.3:* Integrar cambios de estado del recurso en la base de datos.
- **Tarea 3: Notificación y Registro de Auditoría**

    - *Subtarea 3.1:* Implementar notificaciones automáticas cuando un recurso vuelve a estar operativo.
    - *Subtarea 3.2:* Registrar en auditoría las acciones realizadas en cada mantenimiento.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias y de integración con Jasmine.
    - *Subtarea 4.2:* Documentar el flujo de registro de mantenimiento correctivo.

MUST - Módulo de Disponibilidad y Reservas

RF-7: Definir horarios disponibles y validar la disponibilidad en tiempo real, garantizando que no se generen conflictos en las reservas

HU-09: Configurar Horarios Disponibles para Recursos

Historia de Usuario

Como **Administrador**, quiero **definir los horarios disponibles para cada recurso con restricciones institucionales** para asegurar que las reservas se realicen únicamente en periodos autorizados, evitando conflictos y garantizando el cumplimiento de las normativas institucionales.

Criterios de Aceptación

- El sistema debe permitir configurar franjas horarias de disponibilidad para cada recurso (por ejemplo, de 8:00 a 18:00).
- Se deben poder definir restricciones adicionales, como días no operativos, períodos bloqueados por eventos o mantenimientos, y excepciones especiales.
- La configuración debe validar que no existan solapamientos entre franjas horarias ni conflictos con restricciones predefinidas.
- Los cambios en la configuración se deben registrar en el historial de auditoría (usuario, fecha, modificaciones realizadas).
- La vista de disponibilidad de cada recurso debe reflejar claramente las franjas autorizadas y bloqueadas.
- La integración con el proceso de reserva debe impedir que se realicen reservas fuera de los horarios configurados.

Para abordar de forma integral este requerimiento, se desglosa en dos sub-historias:

SubHU-09.1: Configurar Franjas Horarias Básicas

Historia de Usuario

Como **Administrador**, quiero **configurar las franjas horarias básicas de disponibilidad para cada recurso** para que se establezcan los períodos en los que éstos pueden ser reservados de forma normal.

Criterios de Aceptación

- El formulario de configuración debe permitir ingresar la hora de inicio y fin de la disponibilidad (por ejemplo, de 8:00 a 18:00).
- Se debe validar que la hora de inicio sea anterior a la hora de fin.
- Los datos ingresados deben visualizarse en la vista de detalles del recurso.
- Cualquier modificación se debe registrar en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Configuración de Franjas Horarias**
  - *Subtarea 1.1:* Crear wireframes y mockups del formulario para definir las franjas horarias.
  - *Subtarea 1.2:* Validar el diseño con stakeholders y ajustar según retroalimentación.
- **Tarea 2: Desarrollo del Endpoint para Configuración de Horarios en NestJS**
  - *Subtarea 2.1:* Definir el DTO para la configuración de horarios (hora de inicio y fin).
  - *Subtarea 2.2:* Implementar la lógica en el servicio para almacenar y actualizar las franjas horarias.
  - *Subtarea 2.3:* Integrar validaciones que aseguren que la hora de inicio sea anterior a la de fin.
  - *Subtarea 2.4:* Conectar el endpoint con la base de datos mediante Prisma y MongoDB.
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
  - *Subtarea 3.1:* Configurar el log (por ejemplo, con Winston) para registrar cada acción sobre la configuración horaria.
  - *Subtarea 3.2:* Implementar manejo de errores que notifique al usuario si se ingresan datos inválidos.
- **Tarea 4: Pruebas y Documentación**
  - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración con Jasmine utilizando escenarios Given-When-Then.
  - *Subtarea 4.2:* Documentar la funcionalidad en la guía del usuario y actualizar la documentación técnica.
  - *Subtarea 4.3:* Integrar el módulo en el pipeline de CI/CD (GitHub Actions, SonarQube).

SubHU-09.2: Configurar Restricciones Institucionales y Bloqueos

Historia de Usuario

Como **Administrador**, quiero **configurar restricciones institucionales y bloquear ciertos períodos** para que se impida la reserva de recursos durante días no operativos, eventos especiales o mantenimientos, garantizando el cumplimiento de las políticas institucionales.

Criterios de Aceptación

- El sistema debe permitir definir períodos bloqueados especificando fecha, hora de inicio y fin.
- Se debe validar que los períodos bloqueados no se superpongan con las franjas horarias definidas.
- La interfaz debe mostrar visualmente (por ejemplo, en un calendario o lista) los períodos bloqueados asociados a cada recurso.
- Durante la reserva, el sistema debe rechazar solicitudes que coincidan con períodos bloqueados.
- Los cambios en las restricciones deben registrarse en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz para Restricciones y Bloqueos**
    - *Subtarea 1.1:* Crear prototipos (wireframes/mockups) de la pantalla para definir y visualizar períodos bloqueados.
    - *Subtarea 1.2:* Recoger feedback de usuarios clave (administradores y responsables de reservas).
- **Tarea 2: Desarrollo del Módulo de Restricciones en NestJS**
    - *Subtarea 2.1:* Definir el DTO y modelo de datos para los períodos bloqueados (fecha, hora de inicio y fin).
    - *Subtarea 2.2:* Implementar la lógica en el servicio para agregar, editar y eliminar períodos bloqueados.
    - *Subtarea 2.3:* Desarrollar validaciones para evitar solapamientos con las franjas horarias básicas y otros períodos bloqueados.
- **Tarea 3: Integración con el Proceso de Reserva**
    - *Subtarea 3.1:* Modificar la lógica de validación de reservas para comprobar que la solicitud no se realice en períodos bloqueados.
    - *Subtarea 3.2:* Desarrollar pruebas de integración que simulen reservas durante períodos bloqueados y verifiquen el rechazo adecuado.
- **Tarea 4: Registro de Auditoría y Notificaciones**
    - *Subtarea 4.1:* Implementar el registro en el historial de auditoría para cada acción de configuración de restricciones.
    - *Subtarea 4.2:* Configurar notificaciones automáticas para informar a los administradores de cambios críticos en los períodos bloqueados.
- **Tarea 5: Pruebas y Documentación**
    - *Subtarea 5.1:* Escribir pruebas unitarias y de integración (usando Jasmine y escenarios BDD).
    - *Subtarea 5.2:* Documentar la funcionalidad y actualizar la guía del usuario.
    - *Subtarea 5.3:* Integrar esta funcionalidad en el pipeline de CI/CD.

RF-8: Integración con calendarios para sincronizar reservas y evitar solapamientos con eventos institucionales

HU-10: Integración con Calendarios para Evitar Conflictos

Historia de Usuario

Como **Administrador**, quiero **integrar la plataforma con los calendarios institucionales (por ejemplo, Google Calendar, Outlook, iCal)** para sincronizar eventos y evitar conflictos en las reservas de recursos, asegurando que los espacios solo se reserven cuando estén realmente disponibles.

Criterios de Aceptación

- El sistema debe permitir configurar la conexión con al menos un sistema de calendario mediante API (por ejemplo, Google Calendar).

- Se debe disponer de una interfaz para ingresar y validar las credenciales de acceso y configuración de sincronización (URL, claves, tokens, etc.).
- Al iniciar una sincronización, el sistema debe obtener los eventos programados en el calendario institucional y mapearlos a los recursos correspondientes.
- Durante el proceso de reserva, el sistema debe validar en tiempo real que el recurso no tenga un evento en conflicto en el calendario.
- Si se detecta un conflicto, se debe mostrar una notificación clara al usuario indicando la causa para que el usuario decida.
- Toda acción de integración (configuración, sincronización y validación de conflictos) debe registrarse en el historial de auditoría, incluyendo usuario, fecha y detalles de la acción.

Para abordar este requerimiento, se desglosa en dos sub-historias:

SubHU-10.1: Configuración de la Integración con Calendarios

Historia de Usuario

Como **Administrador**, quiero **configurar la conexión con los sistemas de calendario institucionales** para establecer la comunicación necesaria que permita la sincronización de eventos y la validación de disponibilidad.

Criterios de Aceptación

- La interfaz debe permitir ingresar datos de conexión: API Key, Client ID, Secret, URL del endpoint, etc.
- Se debe validar la conexión mediante una prueba de conexión o autenticación y mostrar un mensaje de éxito o error.
- Los datos de configuración deben guardarse de forma segura y poder editarse posteriormente.
- La configuración debe quedar registrada en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Configuración**
  - *Subtarea 1.1:* Crear wireframes y mockups de la pantalla de configuración de la integración con calendarios.
  - *Subtarea 1.2:* Validar el diseño con stakeholders y recoger feedback para ajustes.
- **Tarea 2: Desarrollo del Módulo de Configuración en NestJS**
  - *Subtarea 2.1:* Definir el DTO y modelo de datos para almacenar las credenciales y parámetros de conexión.
  - *Subtarea 2.2:* Implementar el endpoint para guardar y actualizar la configuración.
  - *Subtarea 2.3:* Desarrollar la lógica para realizar una prueba de conexión y autenticación con el sistema de calendario.
  - *Subtarea 2.4:* Registrar en la base de datos la configuración utilizando Prisma y MongoDB.
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**

    - *Subtarea 3.1:* Configurar log (usando Winston) para registrar cada acción de configuración.
    - *Subtarea 3.2:* Implementar manejo de errores y notificaciones en caso de fallos en la conexión.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Escribir pruebas unitarias e integración en Jasmine (utilizando escenarios Given-When-Then) para la configuración.
    - *Subtarea 4.2:* Documentar la funcionalidad, incluyendo ejemplos de configuración y solución de errores.
    - *Subtarea 4.3:* Integrar este módulo en el pipeline de CI/CD.

SubHU-10.2: Sincronización y Validación de Conflictos con Eventos del Calendario

Historia de Usuario

Como **Administrador**, quiero **sincronizar en tiempo real los eventos del calendario institucional y validar que no existan conflictos con las reservas de recursos** para evitar asignaciones erróneas y garantizar que los espacios estén realmente disponibles.

Criterios de Aceptación

- Al iniciar la sincronización, el sistema debe obtener los eventos del calendario y asociarlos al recurso correspondiente, basándose en identificadores o mapeos predefinidos.
- La sincronización debe ejecutarse en un intervalo configurable (por ejemplo, cada 30 minutos) o manualmente mediante un botón de “Sincronizar ahora”.
- Durante el proceso de reserva, el sistema debe consultar los eventos sincronizados y, si hay un conflicto (por ejemplo, el recurso está ocupado en el calendario), notificar al usuario y si el usuario no decide reservar impedir la reserva.
- Se deben manejar adecuadamente las excepciones en caso de fallos en la sincronización o problemas de comunicación con el API del calendario.
- La sincronización y las validaciones deben quedar registradas en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Desarrollo del Conector de Sincronización**
    - *Subtarea 1.1:* Definir el modelo de datos y el mapeo entre eventos del calendario y recursos.
    - *Subtarea 1.2:* Implementar en NestJS un conector que consuma la API del calendario y extraiga los eventos.
    - *Subtarea 1.3:* Configurar la sincronización automática (intervalo configurable) y opción de sincronización manual.
- **Tarea 2: Implementación de la Lógica de Validación de Conflictos**
    - *Subtarea 2.1:* Desarrollar la lógica en el servicio de reservas para consultar los eventos sincronizados.
    - *Subtarea 2.2:* Implementar validaciones que impidan la reserva si hay un evento en conflicto.

    - *Subtarea 2.3:* Mostrar notificaciones claras al usuario sobre el conflicto y sugerir horarios alternativos, si es posible.
- **Tarea 3: Registro de Auditoría y Manejo de Errores**
    - *Subtarea 3.1:* Configurar el registro de auditoría para cada acción de sincronización y validación.
    - *Subtarea 3.2:* Implementar manejo de excepciones que capture y notifique errores en la sincronización.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Escribir pruebas unitarias e integración con Jasmine, simulando escenarios de sincronización y conflicto.
    - *Subtarea 4.2:* Documentar la funcionalidad de sincronización, incluyendo instrucciones para la configuración y resolución de conflictos.
    - *Subtarea 4.3:* Incluir la integración en el pipeline de CI/CD.

RF-10: Visualización de la disponibilidad en formato calendario, imprescindible para la toma de decisiones por parte de los usuarios

HU-11: Visualización de Disponibilidad en Formato Calendario

Historia de Usuario

Como **Usuario**, quiero **visualizar la disponibilidad de los recursos en un calendario interactivo** para planificar mis reservas de manera eficiente y evitar conflictos en la asignación de espacios.

Criterios de Aceptación

- La vista de calendario debe ofrecer opciones para visualizar en formato diario, semanal y mensual.
- Los horarios disponibles y ocupados deben diferenciarse claramente mediante colores o etiquetas.
- Se debe permitir filtrar la visualización por tipo de recurso, ubicación y fecha.
- La información en el calendario se actualizará en tiempo real cuando se realice una reserva, cancelación o modificación.
- La interfaz será responsiva para dispositivos móviles y de escritorio.
- Los cambios realizados en el calendario (por ejemplo, inicio de reserva) deben registrarse en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz del Calendario**
    - *Subtarea 1.1:* Crear wireframes y mockups para las vistas diario, semanal y mensual.
    - *Subtarea 1.2:* Definir la diferenciación visual (colores, etiquetas) para horarios disponibles y ocupados.
    - *Subtarea 1.3:* Incluir opciones de filtrado y validarlas con stakeholders.
- **Tarea 2: Desarrollo del Componente de Calendario en el Frontend**

    - *Subtarea 2.1:* Seleccionar o desarrollar una librería de calendario (por ejemplo, FullCalendar).
    - *Subtarea 2.2:* Integrar el componente en la aplicación y adaptar la visualización a dispositivos móviles.
    - *Subtarea 2.3:* Configurar la comunicación con el backend para cargar la disponibilidad en tiempo real.
- **Tarea 3: Desarrollo del Endpoint para Consulta de Disponibilidad**
    - *Subtarea 3.1:* Definir el DTO para la consulta de disponibilidad (incluyendo filtros por fecha, tipo y ubicación).
    - *Subtarea 3.2:* Implementar la lógica en el servicio de reservas para recuperar y formatear la información.
    - *Subtarea 3.3:* Integrar el endpoint con la base de datos mediante Prisma y MongoDB.
    - *Subtarea 3.4:* Incluir validaciones y manejo de excepciones para consultas inválidas.
- **Tarea 4: Integración en Tiempo Real y Actualización Automática**
    - *Subtarea 4.1:* Implementar mecanismos (como suscripciones o polling) para actualizar el calendario en tiempo real.
    - *Subtarea 4.2:* Conectar el flujo de reservas con la actualización del componente de calendario.
    - *Subtarea 4.3:* Realizar pruebas de integración que verifiquen la actualización automática.
- **Tarea 5: Registro de Auditoría, Pruebas y Documentación**
    - *Subtarea 5.1:* Configurar logging (por ejemplo, con Winston) para registrar acciones realizadas desde la vista de calendario.
    - *Subtarea 5.2:* Desarrollar pruebas unitarias e integración utilizando Jasmine y escenarios BDD (Given-When-Then).
    - *Subtarea 5.3:* Documentar la funcionalidad en la guía del usuario y la documentación técnica.
    - *Subtarea 5.4:* Integrar el módulo en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

SubHU-11.1: Interacción para Iniciar Reservas desde el Calendario

Historia de Usuario

Como **Usuario con Permiso de Reserva**, quiero **iniciar una reserva directamente desde el calendario** para agilizar el proceso y asegurar que el recurso se reserve en un horario disponible.

Criterios de Aceptación

- Al hacer clic en una franja horaria disponible, se debe abrir un formulario de reserva pre-llenado con la información del recurso y la franja seleccionada.
- El sistema debe validar en tiempo real la disponibilidad del recurso antes de confirmar la acción.
- Si existe algún conflicto, se mostrará una notificación clara y se sugerirán alternativas.

- La acción de iniciar una reserva desde el calendario debe registrarse en el historial de auditoría.
- Si la reserva requiere validación, el flujo se integrará con el módulo de aprobaciones.

Tareas y Subtareas

- **Tarea 1: Diseño del Flujo de Interacción en el Calendario**
  - *Subtarea 1.1:* Crear mockups que muestren el proceso de clic en una franja disponible y apertura del formulario de reserva.
  - *Subtarea 1.2:* Validar el diseño con usuarios y administradores.
- **Tarea 2: Desarrollo del Componente de Reserva Directa**
  - *Subtarea 2.1:* Implementar en el frontend la captura del evento de clic sobre una franja.
  - *Subtarea 2.2:* Desarrollar el formulario de reserva pre-llenado y conectarlo con el endpoint de creación de reservas.
  - *Subtarea 2.3:* Asegurar que se realicen validaciones en tiempo real sobre la disponibilidad.
- **Tarea 3: Integración con el Módulo de Reservas y Validación**
  - *Subtarea 3.1:* Ajustar la lógica de reserva para incluir la validación del calendario.
  - *Subtarea 3.2:* Implementar manejo de errores y notificaciones en caso de conflicto.
  - *Subtarea 3.3:* Desarrollar pruebas de integración que simulen el flujo completo de reserva iniciada desde el calendario.
- **Tarea 4: Registro de Auditoría y Documentación**
  - *Subtarea 4.1:* Configurar el registro de auditoría para capturar todas las acciones de reserva iniciadas desde el calendario.
  - *Subtarea 4.2:* Escribir escenarios BDD en Jasmine y realizar pruebas de aceptación.
  - *Subtarea 4.3:* Documentar el proceso y actualizar la guía del usuario.

RF-11: Registro del historial de uso de cada recurso, fundamental para la trazabilidad

HU-12: Registro del Historial de Uso de Recursos

Historia de Usuario

Como **Administrador o Usuario Autorizado**, quiero **registrar y consultar el historial de uso de cada recurso** para auditar las reservas, analizar patrones de uso y optimizar la planificación y asignación de recursos.

Criterios de Aceptación

- El sistema debe registrar automáticamente cada acción sobre reservas (creación, modificación, cancelación) para cada recurso.
- Cada registro debe incluir:
  - Identificación del usuario que realizó la acción.

    - Fecha y hora de la acción.
    - Duración de la reserva.
    - Estado de la reserva (confirmada, cancelada, completada).
    - Descripción o incidencias asociadas (si las hubiera).
- La información registrada debe ser inalterable y estar disponible para consulta.
- La interfaz de consulta debe permitir filtrar registros por recurso, usuario, fecha y estado.
- La consulta debe responder en menos de 2 segundos en condiciones normales.
- Se debe permitir exportar el historial en formato CSV.
- Todas las acciones de registro y consulta deben quedar registradas en el historial de auditoría.

Para cubrir todos estos aspectos, se desglosa en dos sub-historias:

SubHU-12.1: Registro Automático del Historial de Reservas

Historia de Usuario

Como **Administrador**, quiero **que el sistema registre automáticamente cada acción sobre reservas** para contar con un historial completo y verificable de la utilización de cada recurso.

Criterios de Aceptación

- Cada acción (creación, modificación o cancelación) se registra automáticamente en el historial.
- Los registros incluyen: usuario, fecha, hora, recurso, duración, estado de la reserva y cualquier incidencia.
- Los registros se almacenan de forma inalterable para fines de auditoría.
- Se notifica al administrador en caso de error crítico durante el registro.

Tareas y Subtareas

- **Tarea 1: Diseño del Modelo de Datos del Historial**
    - *Subtarea 1.1:* Definir el esquema del historial de reservas (campos, tipos de datos).
    - *Subtarea 1.2:* Validar el diseño con el equipo de desarrollo y auditoría.
- **Tarea 2: Desarrollo del Módulo de Registro Automático en NestJS**
    - *Subtarea 2.1:* Implementar un middleware o interceptor para capturar acciones sobre reservas.
    - *Subtarea 2.2:* Crear la lógica en el servicio para almacenar los registros en la base de datos (MongoDB con Prisma).
    - *Subtarea 2.3:* Implementar validaciones para asegurar la integridad de los registros.
- **Tarea 3: Configuración de Auditoría y Manejo de Excepciones**
    - *Subtarea 3.1:* Configurar herramientas de logging (por ejemplo, Winston) para registrar cada acción.
    - *Subtarea 3.2:* Implementar manejo de excepciones para capturar y notificar fallos en el registro.

- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine y escenarios BDD (Given-When-Then).
    - *Subtarea 4.2:* Documentar el modelo de datos, el flujo de registro y los criterios de auditoría.
    - *Subtarea 4.3:* Integrar el módulo en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

SubHU-12.2: Consulta y Visualización del Historial de Uso

Historia de Usuario

Como **Administrador o Usuario Autorizado**, quiero **consultar y visualizar el historial de uso de cada recurso** para auditar las reservas y detectar patrones que permitan optimizar la asignación de recursos.

Criterios de Aceptación

- La interfaz debe mostrar una lista detallada con: usuario, fecha, hora, recurso, duración, estado y descripción de incidencias.
- Debe permitirse filtrar el historial por recurso, usuario, fecha y estado.
- La consulta debe responder en menos de 2 segundos en condiciones normales.
- Se debe ofrecer la opción de exportar el historial filtrado en formato CSV.
- La información mostrada debe corresponder al registro automático y no poder modificarse.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Consulta del Historial**
    - *Subtarea 1.1:* Crear wireframes y mockups de la pantalla de historial, incluyendo filtros y opción de exportación.
    - *Subtarea 1.2:* Validar el diseño con administradores y usuarios autorizados.
- **Tarea 2: Desarrollo del Endpoint de Consulta en NestJS**
    - *Subtarea 2.1:* Definir el DTO para la consulta del historial (incluyendo filtros).
    - *Subtarea 2.2:* Implementar la lógica en el servicio para recuperar y formatear los registros.
    - *Subtarea 2.3:* Optimizar la consulta para garantizar tiempos de respuesta adecuados.
- **Tarea 3: Funcionalidad de Exportación**
    - *Subtarea 3.1:* Implementar la exportación de los registros filtrados a formato CSV.
    - *Subtarea 3.2:* Integrar el endpoint de exportación en la interfaz de usuario.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración con Jasmine y escenarios BDD.
    - *Subtarea 4.2:* Documentar el proceso de consulta, filtrado y exportación del historial.
    - *Subtarea 4.3:* Integrar la funcionalidad en el pipeline de CI/CD.

Should - Módulo de Disponibilidad y Reservas

RF-12: Permitir reservas periódicas, facilitando la planificación a largo plazo de espacios

HU-13: Permitir Reservas Periódicas

Historia de Usuario

Como **Usuario**, quiero **realizar reservas periódicas** para programar de forma recurrente el uso de un recurso, ahorrando tiempo y facilitando la planificación a largo plazo.

Criterios de Aceptación

- En el formulario de reserva debe existir la opción de habilitar la reserva periódica.
- Se deben permitir definir la fecha de inicio y la fecha de fin de la recurrencia.
- Debe permitirse seleccionar la frecuencia de repetición (diaria, semanal o mensual).
- El sistema debe generar una vista previa de todas las fechas en las que se aplicará la reserva periódica.
- Se debe validar la disponibilidad de cada instancia generada; en caso de conflicto, se notificará al usuario.
- El sistema debe permitir modificar o cancelar la serie completa o instancias individuales.
- Todas las acciones (creación, modificación o cancelación) deben quedar registradas en el historial de auditoría.

Para cubrir todos los aspectos del requerimiento se desglosa en dos sub-historias:

SubHU-13.1: Configurar Reserva Periódica

Historia de Usuario

Como **Usuario**, quiero **configurar una reserva periódica al momento de realizar una solicitud** para que el sistema genere automáticamente las instancias de reserva en el rango y con la frecuencia seleccionada.

Criterios de Aceptación

- El formulario de reserva debe incluir un campo (checkbox o switch) para activar la opción "Reserva periódica".
- Al activarla, el formulario deberá mostrar campos para ingresar la fecha de inicio, fecha de fin y seleccionar la frecuencia (diaria, semanal, mensual).
- El sistema deberá mostrar una vista previa con la lista de fechas generadas.
- Se validará que la fecha de inicio sea anterior a la fecha de fin y que las fechas generadas cumplan con la disponibilidad del recurso.
- La configuración de la reserva periódica se guardará junto con la reserva.

- La acción de crear una reserva periódica quedará registrada en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Reserva Periódica**
  - *Subtarea 1.1:* Crear wireframes y mockups del formulario de reserva que incluya la opción de reserva periódica y campos para fecha de inicio, fecha de fin y frecuencia.
  - *Subtarea 1.2:* Validar el diseño con usuarios y stakeholders, recopilando retroalimentación para ajustes.
- **Tarea 2: Desarrollo del Endpoint de Configuración de Recurrencia**
  - *Subtarea 2.1:* Definir el DTO para incluir datos de recurrencia (fecha inicio, fecha fin, frecuencia).
  - *Subtarea 2.2:* Implementar la lógica en el servicio de reservas para generar automáticamente las fechas de reserva según la frecuencia seleccionada.
  - *Subtarea 2.3:* Realizar validaciones: que la fecha de inicio sea anterior a la fecha de fin, que la frecuencia generada no genere conflictos con la disponibilidad.
  - *Subtarea 2.4:* Integrar el endpoint con la base de datos usando Prisma y MongoDB.
- **Tarea 3: Registro de Auditoría y Manejo de Errores**
  - *Subtarea 3.1:* Configurar logging (usando Winston u otra herramienta) para registrar la creación de reservas periódicas.
  - *Subtarea 3.2:* Implementar manejo de excepciones para capturar errores en el proceso de generación de fechas y notificar al usuario.
- **Tarea 4: Pruebas y Documentación**
  - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración con Jasmine utilizando escenarios Given-When-Then para el flujo de reserva periódica.
  - *Subtarea 4.2:* Documentar la funcionalidad en la guía del usuario y actualizar la documentación técnica.
  - *Subtarea 4.3:* Integrar la funcionalidad en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

SubHU-13.2: Modificar y Cancelar Reservas Periódicas

Historia de Usuario

Como **Usuario**, quiero **modificar o cancelar reservas periódicas** para ajustar mi serie de reservas según cambios en mis necesidades sin afectar las instancias ya confirmadas.

Criterios de Aceptación

- El sistema debe permitir visualizar la serie completa de reservas periódicas en el panel del usuario.
- Se debe ofrecer la opción de modificar o cancelar la serie completa o una única instancia.
- Al modificar la serie, se debe validar nuevamente la disponibilidad para las nuevas fechas.

- La cancelación de una instancia individual debe reflejarse en la vista sin afectar el resto de la serie.
- Todas las modificaciones y cancelaciones deben quedar registradas en el historial de auditoría, indicando el usuario, fecha y cambios realizados.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz para Gestión de Reservas Periódicas**
    - *Subtarea 1.1:* Crear wireframes y mockups de la vista de series de reservas periódicas, incluyendo opciones de edición y cancelación.
    - *Subtarea 1.2:* Validar el diseño con stakeholders y ajustar el flujo según requerimientos.
- **Tarea 2: Desarrollo del Módulo de Modificación/Cancellation de Reservas Periódicas**
    - *Subtarea 2.1:* Definir el DTO para la modificación y cancelación de reservas periódicas.
    - *Subtarea 2.2:* Implementar la lógica en el servicio de reservas para permitir la modificación de la serie o de una instancia individual.
    - *Subtarea 2.3:* Integrar validaciones que aseguren que los cambios no generen conflictos de disponibilidad.
    - *Subtarea 2.4:* Actualizar el modelo de datos para reflejar el estado de cada instancia (activa, cancelada, modificada).
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
    - *Subtarea 3.1:* Configurar el registro de auditoría para capturar todas las acciones de modificación y cancelación.
    - *Subtarea 3.2:* Implementar manejo de excepciones que notifique al usuario en caso de error durante la actualización.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Escribir pruebas unitarias e integración (BDD con Jasmine) para validar el flujo de modificación y cancelación.
    - *Subtarea 4.2:* Documentar el proceso y actualizar la guía del usuario.
    - *Subtarea 4.3:* Integrar el módulo en el pipeline de CI/CD.

RF-14: Implementar lista de espera para reservas sobrecargadas, para maximizar la utilización de los recursos

HU-14: Gestión de Lista de Espera para Reservas Sobrecargadas

Historia de Usuario

Como **Usuario**, quiero **unirme a la lista de espera cuando no haya disponibilidad para reservar un recurso** para tener la oportunidad de obtenerlo tan pronto se libere, sin tener que monitorear constantemente su disponibilidad.

Criterios de Aceptación

- Cuando un usuario intenta reservar un recurso sin disponibilidad, se le debe ofrecer la opción de unirse a una lista de espera.

- El sistema debe registrar automáticamente la fecha y hora de inscripción, asignando un orden basado en el momento de ingreso.
- La interfaz debe permitir al usuario visualizar su posición en la lista de espera.
- El usuario debe tener la opción de cancelar su inscripción en cualquier momento.
- Todas las acciones (inscripción, actualización o cancelación) deben registrarse en el historial de auditoría.
- La lógica de la lista de espera debe integrarse con el proceso de asignación: cuando se libera un recurso, el primer usuario en la lista es notificado con un tiempo limitado para confirmar la reserva.

Para abarcar de forma completa este requerimiento, se desglosa en dos sub-historias:

SubHU-14.1: Inscripción en la Lista de Espera

Historia de Usuario

Como **Usuario**, quiero **inscribirme en la lista de espera cuando un recurso esté completamente reservado** para asegurar mi posición y tener la oportunidad de ser notificado cuando el recurso se libere.

Criterios de Aceptación

- Al intentar reservar un recurso no disponible, el sistema debe ofrecer la opción "Unirse a la lista de espera".
- El usuario debe poder confirmar su inscripción y el sistema debe registrar automáticamente la fecha, hora y asignar un número de orden.
- La interfaz mostrará la posición actual del usuario en la lista de espera.
- Se debe permitir la cancelación de la inscripción en cualquier momento.
- La inscripción y cualquier cancelación deben quedar registradas en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Inscripción**
  - *Subtarea 1.1:* Crear wireframes y mockups del flujo de inscripción en la lista de espera.
  - *Subtarea 1.2:* Validar el diseño con usuarios y administradores para asegurar claridad y facilidad de uso.
- **Tarea 2: Desarrollo del Endpoint para Inscripción**
  - *Subtarea 2.1:* Definir el DTO para la inscripción en la lista de espera, incluyendo datos del usuario, recurso y timestamp.
  - *Subtarea 2.2:* Implementar la lógica en el servicio de reservas para agregar el usuario a la lista de espera, asegurando que no se duplique la inscripción para el mismo recurso.
  - *Subtarea 2.3:* Integrar el endpoint con la base de datos (MongoDB usando Prisma).
- **Tarea 3: Registro de Auditoría y Manejo de Errores**
  - *Subtarea 3.1:* Configurar logging (por ejemplo, con Winston) para registrar cada acción de inscripción o cancelación.

    - *Subtarea 3.2:* Implementar manejo de excepciones para notificar al usuario en caso de error.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración en Jasmine utilizando escenarios BDD (Given-When-Then).
    - *Subtarea 4.2:* Documentar el proceso de inscripción y las reglas de la lista de espera en la guía del usuario.
    - *Subtarea 4.3:* Incluir esta funcionalidad en el pipeline de CI/CD.

SubHU-14.2: Notificación y Confirmación de Reserva desde la Lista de Espera

Historia de Usuario

Como **Usuario en la Lista de Espera**, quiero **ser notificado cuando un recurso se libere y disponer de un tiempo limitado para confirmar mi reserva** para asegurar mi acceso al recurso antes de que otro usuario tome mi lugar.

Criterios de Aceptación

- Cuando un recurso se libera (por cancelación o modificación), el sistema debe notificar automáticamente al primer usuario en la lista de espera.
- La notificación debe incluir un enlace o acción que permita confirmar la reserva y un tiempo configurable (por ejemplo, 10 minutos) para hacerlo.
- Si el usuario confirma dentro del plazo, la reserva se asigna; de lo contrario, el sistema debe pasar al siguiente usuario en la lista.
- La confirmación o expiración de la notificación se registrará en el historial de auditoría.
- Se deben enviar recordatorios si el tiempo de confirmación está por expirar.

Tareas y Subtareas

- **Tarea 1: Desarrollo del Módulo de Notificaciones para la Lista de Espera**
    - *Subtarea 1.1:* Implementar la lógica para detectar cuando un recurso se libera y obtener el primer usuario en la lista.
    - *Subtarea 1.2:* Desarrollar el formato de notificación (correo electrónico, mensaje en la app, o WhatsApp) que incluya detalles del recurso y el enlace para confirmar.
    - *Subtarea 1.3:* Configurar el tiempo límite para la confirmación y establecer recordatorios automáticos.
- **Tarea 2: Desarrollo de la Lógica de Confirmación y Escalado**
    - *Subtarea 2.1:* Implementar la lógica en el servicio de reservas para aceptar la confirmación del usuario y asignar el recurso.
    - *Subtarea 2.2:* Si el usuario no confirma en el tiempo estipulado, activar la notificación al siguiente usuario en la lista.
    - *Subtarea 2.3:* Registrar todas las acciones (confirmación, expiración, escalado) en la base de datos.
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
    - *Subtarea 3.1:* Configurar logging para registrar cada notificación y acción de confirmación.

    - *Subtarea 3.2:* Implementar manejo de errores que notifique a los administradores en caso de fallos en el proceso.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Escribir pruebas unitarias e integración con Jasmine utilizando escenarios Given-When-Then.
    - *Subtarea 4.2:* Documentar el flujo de notificación y confirmación, incluyendo la configuración del tiempo límite.
    - *Subtarea 4.3:* Integrar la funcionalidad en el pipeline de CI/CD.

RF-15: Reasignación de reservas en caso de mantenimiento o eventos imprevistos, considerando dependencias funcionales

HU-15: Reasignación de Reservas en Caso de Mantenimiento o Eventos Imprevistos

Historia de Usuario

Como **Administrador**, quiero **reasignar reservas automáticamente o de forma manual cuando un recurso se vuelva inactivo por mantenimiento o eventos imprevistos** para minimizar la interrupción del servicio y asegurar que los usuarios afectados obtengan una alternativa o sean notificados para tomar una decisión.

Criterios de Aceptación Generales

- Cuando un recurso se vuelve inactivo (por mantenimiento, falla técnica o evento imprevisto), el sistema debe identificar todas las reservas afectadas y cambiar su estado a "Pendiente de Reasignación".
- Si hay recursos alternativos disponibles, el sistema debe intentar reasignar automáticamente la reserva y notificar al usuario afectado.
- En caso de que la reasignación automática no sea posible o el usuario rechace la opción, el sistema debe permitir la intervención manual por parte del administrador.
- Las notificaciones automáticas y manuales deben incluir un tiempo límite configurable para que el usuario confirme la nueva asignación.
- Todas las acciones (automáticas y manuales) deben quedar registradas en el historial de auditoría, indicando usuario, fecha, recurso asignado y acción realizada.
- La funcionalidad debe integrarse con el módulo de reservas y con la gestión de disponibilidad en tiempo real.

Para cubrir los distintos flujos del RF-15 se desglosa en dos sub-historias:

SubHU-15.1: Reasignación Automática de Reservas

Historia de Usuario

Como **Administrador**, quiero **que el sistema reasigne automáticamente las reservas afectadas cuando un recurso se vuelva inactivo por mantenimiento o eventos**

imprevistos para reducir el tiempo de inactividad y optimizar la utilización de recursos sin intervención manual inmediata.

Criterios de Aceptación

- El sistema detecta automáticamente el cambio de estado del recurso a "No disponible" (por mantenimiento o imprevistos).
- Las reservas asociadas se actualizan a un estado "Pendiente de Reasignación" y se dispara el proceso de búsqueda de recursos alternativos equivalentes.
- Si se encuentra un recurso alternativo disponible, el sistema reasigna la reserva y notifica al usuario.
- Si se produce algún conflicto durante la reasignación (por ejemplo, disponibilidad simultánea), el sistema notifica al usuario para que elija entre opciones alternativas o active el flujo manual.
- Todas las acciones y resultados deben registrarse en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Detección Automática del Cambio de Estado del Recurso**
  - *Subtarea 1.1:* Implementar lógica en el servicio de reservas para detectar cuando un recurso cambia su estado a “No disponible”.
  - *Subtarea 1.2:* Integrar esta lógica con el módulo de mantenimiento o de eventos imprevistos.
  - *Subtarea 1.3:* Desarrollar pruebas unitarias para verificar la detección correcta.
- **Tarea 2: Desarrollo del Algoritmo de Reasignación Automática**
  - *Subtarea 2.1:* Definir el algoritmo para seleccionar recursos alternativos equivalentes.
  - *Subtarea 2.2:* Implementar la lógica en el servicio de reservas para actualizar automáticamente el estado de la reserva a “Pendiente de Reasignación” y reasignar si es posible.
  - *Subtarea 2.3:* Validar la disponibilidad del recurso alternativo antes de confirmar la reasignación.
  - *Subtarea 2.4:* Notificar al usuario afectado con los detalles de la reasignación y el tiempo límite para confirmar.
  - *Subtarea 2.5:* Escribir pruebas unitarias e integración (BDD con Jasmine, Given-When-Then).
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
  - *Subtarea 3.1:* Configurar logging (usando Winston) para registrar cada acción automática de reasignación.
  - *Subtarea 3.2:* Implementar manejo de excepciones para capturar errores en el proceso y notificar al administrador.
- **Tarea 4: Documentación y Validación**
  - *Subtarea 4.1:* Documentar el proceso de detección y reasignación automática en la guía del usuario.
  - *Subtarea 4.2:* Integrar el módulo en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

SubHU-15.2: Reasignación Manual y Confirmación por Usuario

Historia de Usuario

Como **Administrador**, quiero **poder intervenir manualmente en la reasignación de reservas afectadas y permitir que los usuarios confirmen o rechacen la nueva asignación** para gestionar casos excepcionales donde la reasignación automática no sea viable o requiera ajustes personalizados.

Criterios de Aceptación

- Si la reasignación automática falla o el usuario rechaza la opción automática, el sistema debe permitir la reasignación manual.
- El administrador debe disponer de una interfaz que muestre todas las reservas afectadas y recursos alternativos disponibles.
- El sistema debe enviar notificaciones al usuario con la opción de confirmar o rechazar la nueva asignación.
- Si el usuario confirma, la reserva se actualiza; si rechaza, se notifica la cancelación o se deriva a la lista de espera.
- Todas las acciones realizadas manualmente deben registrarse en el historial de auditoría.
- La funcionalidad debe incluir validaciones para garantizar que el recurso alternativo esté realmente disponible.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Reasignación Manual**
  - *Subtarea 1.1:* Crear wireframes y mockups de la pantalla de reasignación manual para administradores.
  - *Subtarea 1.2:* Definir la presentación de los recursos alternativos disponibles y el estado de cada reserva afectada.
  - *Subtarea 1.3:* Validar el diseño con stakeholders y ajustar según requerimientos.
- **Tarea 2: Desarrollo del Endpoint para Reasignación Manual**
  - *Subtarea 2.1:* Definir el DTO para la actualización manual de reservas.
  - *Subtarea 2.2:* Implementar la lógica en el servicio de reservas para actualizar la reserva con el recurso alternativo seleccionado.
  - *Subtarea 2.3:* Integrar validaciones que aseguren que el recurso alternativo esté disponible.
  - *Subtarea 2.4:* Actualizar el estado de la reserva y registrar la acción.
- **Tarea 3: Desarrollo de la Lógica de Notificación y Confirmación**
  - *Subtarea 3.1:* Implementar la lógica para enviar notificaciones al usuario con la opción de confirmar o rechazar la reasignación manual.
  - *Subtarea 3.2:* Desarrollar el flujo de confirmación: si el usuario confirma, actualizar la reserva; si rechaza, derivar a cancelación o a la lista de espera.
  - *Subtarea 3.3:* Registrar cada acción en el historial de auditoría.
- **Tarea 4: Pruebas y Documentación**
  - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración con Jasmine utilizando escenarios Given-When-Then.

- *Subtarea 4.2:* Documentar el proceso de reasignación manual y el flujo de confirmación de usuario.
- *Subtarea 4.3:* Integrar la funcionalidad en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

Could - Módulo de Disponibilidad y Reservas

RF-9: Funcionalidad avanzada de búsqueda de recursos (por nombre, tipo, ubicación), que puede ser refinada en fases posteriores sin afectar el core

HU-16: Búsqueda Avanzada y Disponibilidad de Recursos

Historia de Usuario

Como **Usuario**, quiero **realizar búsquedas avanzadas de recursos utilizando múltiples criterios (nombre, tipo, ubicación y disponibilidad)** para encontrar rápidamente el recurso que se adapte a mis necesidades y poder reservarlo de forma eficiente.

Criterios de Aceptación

- El sistema debe permitir la búsqueda utilizando uno o más criterios simultáneamente:
    - Nombre del recurso.
    - Tipo (salón, laboratorio, auditorio, equipo, etc.).
    - Ubicación (edificio, piso, etc.).
    - Disponibilidad en un rango de fechas y horas.
- Los resultados de búsqueda deben mostrarse en una interfaz clara, con indicadores visuales del estado (disponible, ocupado, bloqueado).
- La búsqueda debe actualizarse en tiempo real o mediante una acción de “Buscar” rápida (tiempo de respuesta < 2 segundos).
- Se deben incluir opciones de filtrado y ordenamiento (por ejemplo, por disponibilidad o por popularidad).
- La funcionalidad debe integrarse con el módulo de reservas para verificar en tiempo real la disponibilidad del recurso.
- Todas las búsquedas y consultas realizadas deben quedar registradas en el historial de auditoría.

Para abordar el requerimiento de forma integral se desglosa en dos sub-historias:

SubHU-16.1: Implementación de Filtros y Criterios de Búsqueda

Historia de Usuario

Como **Usuario**, quiero **aplicar filtros avanzados (nombre, tipo, ubicación y disponibilidad)** al buscar recursos para que los resultados sean precisos y se adapten a mis requerimientos específicos.

Criterios de Aceptación

- El formulario de búsqueda debe permitir la selección de múltiples criterios simultáneamente.

- Los filtros deben validar que los datos ingresados sean del formato esperado (por ejemplo, fecha en formato correcto, texto en el campo de nombre).
- Al aplicar los filtros, la vista de resultados debe actualizarse mostrando únicamente los recursos que cumplan con los criterios.
- La interfaz debe ofrecer opciones para limpiar o modificar los filtros.
- Se debe registrar la acción de búsqueda con los filtros aplicados en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Búsqueda Avanzada**
  - *Subtarea 1.1:* Crear wireframes y mockups del formulario de búsqueda avanzada que incluya campos para nombre, tipo, ubicación y rango de disponibilidad.
  - *Subtarea 1.2:* Validar el diseño con usuarios y stakeholders, incorporando retroalimentación.
- **Tarea 2: Desarrollo del Endpoint de Búsqueda en NestJS**
  - *Subtarea 2.1:* Definir el DTO que reciba los criterios de búsqueda.
  - *Subtarea 2.2:* Implementar la lógica en el servicio para filtrar los recursos según los criterios proporcionados.
  - *Subtarea 2.3:* Integrar validaciones de datos en el backend (por ejemplo, formato de fechas, campos obligatorios si aplica).
  - *Subtarea 2.4:* Conectar el endpoint con la base de datos utilizando Prisma y MongoDB.
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
  - *Subtarea 3.1:* Configurar el registro de auditoría para almacenar cada consulta de búsqueda y los filtros aplicados.
  - *Subtarea 3.2:* Implementar manejo de excepciones para capturar errores en la búsqueda y notificar adecuadamente al usuario.
- **Tarea 4: Pruebas y Documentación**
  - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración con Jasmine utilizando escenarios BDD (Given-When-Then) para el flujo de búsqueda avanzada.
  - *Subtarea 4.2:* Documentar la funcionalidad de búsqueda avanzada en la guía del usuario y la documentación técnica.
  - *Subtarea 4.3:* Incluir este módulo en el pipeline de CI/CD.

SubHU-16.2: Visualización en Tiempo Real de la Disponibilidad de Recursos

Historia de Usuario

Como **Usuario**, quiero **ver la disponibilidad actualizada en tiempo real de los recursos** para tomar decisiones informadas y evitar conflictos de reserva.

Criterios de Aceptación

- La vista de resultados de búsqueda debe indicar claramente el estado de cada recurso (por ejemplo, disponible, ocupado, bloqueado).

- La información de disponibilidad se debe actualizar en tiempo real o mediante una acción de “refrescar” rápida (tiempo de respuesta < 2 segundos).
- La interfaz debe mostrar indicadores visuales (colores, etiquetas) que diferencien claramente los estados de disponibilidad.
- La integración con el módulo de reservas debe permitir que cualquier cambio (reserva, cancelación, modificación) se refleje inmediatamente en la búsqueda.
- Se debe registrar cada consulta de disponibilidad en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño del Componente de Visualización en Tiempo Real**
    - *Subtarea 1.1:* Crear wireframes y mockups que muestren el estado de disponibilidad de los recursos en la vista de resultados.
    - *Subtarea 1.2:* Definir la simbología visual (colores, íconos, etiquetas) para los diferentes estados.
- **Tarea 2: Desarrollo del Componente de Actualización en Tiempo Real**
    - *Subtarea 2.1:* Seleccionar o desarrollar un mecanismo (por ejemplo, WebSockets, polling) para actualizar los estados en tiempo real.
    - *Subtarea 2.2:* Integrar este mecanismo en el frontend y conectarlo con el backend.
    - *Subtarea 2.3:* Asegurar que la actualización se realice en el tiempo de respuesta requerido (< 2 segundos).
- **Tarea 3: Desarrollo del Endpoint de Consulta de Disponibilidad**
    - *Subtarea 3.1:* Definir el DTO para la consulta de disponibilidad, incluyendo parámetros de filtrado y actualización.
    - *Subtarea 3.2:* Implementar la lógica en el servicio para obtener y formatear la información actual de disponibilidad.
    - *Subtarea 3.3:* Integrar el endpoint con la base de datos y asegurar el manejo de excepciones en caso de fallos.
- **Tarea 4: Registro de Auditoría y Pruebas**
    - *Subtarea 4.1:* Configurar logging para registrar cada consulta de disponibilidad y actualización en tiempo real.
    - *Subtarea 4.2:* Escribir pruebas unitarias e integración (BDD con Jasmine) que simulen cambios en el estado y verifiquen la actualización en tiempo real.
    - *Subtarea 4.3:* Documentar la funcionalidad en la guía del usuario y la documentación técnica.
    - *Subtarea 4.4:* Integrar la funcionalidad en el pipeline de CI/CD.

MUST - Módulo de Aprobaciones y Gestión de Solicitudes

RF-20: Validar solicitudes de reserva por parte de un responsable (director, ingeniero de soporte o secretaria), esencial para la asignación correcta de recursos

HU-17: Validación de Solicitudes de Reserva

Historia de Usuario

Como **Responsable de Validación (Director, Ingeniero de Soporte o Secretaria)**, quiero **revisar y validar las solicitudes de reserva** para asegurar que los recursos se asignen correctamente de acuerdo con las políticas institucionales y evitar conflictos o usos indebidos.

Criterios de Aceptación

- El sistema debe mostrar una lista de solicitudes de reserva pendientes de validación, con información detallada (solicitante, recurso, fecha y hora, motivo de la reserva, etc.).
- El responsable debe poder aprobar, rechazar o solicitar modificaciones en cada solicitud.
- Al aprobar una solicitud, la reserva se asigna y se actualiza el estado a “Confirmada”; al rechazarla, el estado pasa a “Rechazada” y se debe incluir un motivo de rechazo.
- Si se solicita modificación, la solicitud vuelve al solicitante con comentarios y se marca como “Pendiente de Revisión”.
- Cada acción (aprobación, rechazo, solicitud de modificación) debe registrarse en el historial de auditoría (incluyendo el responsable, la fecha, la hora y el detalle de la acción).
- El sistema debe enviar notificaciones automáticas al solicitante informando sobre el estado de su solicitud.

Para cubrir integralmente el requerimiento, se desglosa en dos sub-historias:

SubHU-17.1: Gestión de la Decisión de Validación

Historia de Usuario

Como **Responsable de Validación**, quiero **tomar decisiones (aprobar, rechazar o solicitar modificaciones) sobre las solicitudes de reserva** para asegurar que sólo se asignen recursos cuando se cumplan todos los criterios y normativas.

Criterios de Aceptación

- El responsable debe disponer de botones o acciones en la interfaz para aprobar, rechazar o solicitar modificaciones.

- Al aprobar, la reserva se marca como “Confirmada” y se asigna el recurso; al rechazar, se requiere ingresar un motivo de rechazo.
- En caso de solicitar modificaciones, la solicitud se devuelve al solicitante con comentarios específicos.
- Todas las decisiones deben actualizar el estado de la solicitud y registrarse en el historial de auditoría.
- La interfaz debe refrescar la lista de solicitudes pendientes en tiempo real tras cada acción.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Validación de Solicitudes**
  - *Subtarea 1.1:* Crear wireframes y mockups de la pantalla de validación que muestre la lista de solicitudes con botones para “Aprobar”, “Rechazar” y “Solicitar Modificación”.
  - *Subtarea 1.2:* Validar el diseño con responsables y usuarios clave para asegurar claridad y usabilidad.
- **Tarea 2: Desarrollo del Endpoint para la Gestión de Decisiones**
  - *Subtarea 2.1:* Definir el DTO para enviar la acción de validación (aprobación, rechazo con motivo, solicitud de modificación con comentarios).
  - *Subtarea 2.2:* Implementar la lógica en el servicio de reservas para actualizar el estado de la solicitud según la decisión.
  - *Subtarea 2.3:* Integrar validaciones en el backend para asegurar que, en caso de rechazo, se ingrese un motivo.
  - *Subtarea 2.4:* Registrar cada acción en el historial de auditoría utilizando Prisma y MongoDB.
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
  - *Subtarea 3.1:* Configurar logging (usando Winston) para capturar las acciones de validación.
  - *Subtarea 3.2:* Implementar manejo de excepciones y notificaciones en caso de errores en la actualización de estado.
- **Tarea 4: Pruebas y Documentación**
  - *Subtarea 4.1:* Escribir pruebas unitarias e integración en Jasmine, utilizando escenarios Given-When-Then para cada acción de validación.
  - *Subtarea 4.2:* Documentar el flujo de validación en la guía del usuario y en la documentación técnica.
  - *Subtarea 4.3:* Integrar este módulo en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

SubHU-17.2: Notificación Automática al Solicitante

Historia de Usuario

Como **Responsable de Validación**, quiero **notificar automáticamente al solicitante sobre la decisión tomada en su solicitud de reserva** para que éste reciba información oportuna y pueda tomar las acciones correspondientes (aceptar, modificar o cancelar).

Criterios de Aceptación

- Al finalizar la validación, el sistema debe enviar una notificación (por correo electrónico, notificación en la app o WhatsApp) al solicitante.
- La notificación debe incluir: el estado final de la solicitud (aprobada, rechazada o pendiente de modificación), el motivo en caso de rechazo o los comentarios en caso de solicitud de modificación, y, en caso de aprobación, la confirmación de la asignación.
- La notificación debe enviarse de forma automática inmediatamente después de la acción del responsable.
- La acción de notificación debe quedar registrada en el historial de auditoría.
- Se debe ofrecer la posibilidad al solicitante de consultar el estado de su solicitud en su panel de usuario.

Tareas y Subtareas

- **Tarea 1: Diseño de la Plantilla de Notificación**
  - *Subtarea 1.1:* Crear un diseño de plantilla para notificaciones, adaptable a los diferentes estados de la solicitud.
  - *Subtarea 1.2:* Validar el diseño con stakeholders y ajustar según retroalimentación.
- **Tarea 2: Desarrollo del Módulo de Notificación en NestJS**
  - *Subtarea 2.1:* Definir el DTO para enviar datos de notificación (estado de la solicitud, motivo o comentarios, información del recurso).
  - *Subtarea 2.2:* Implementar la lógica en el servicio para generar y enviar la notificación.
  - *Subtarea 2.3:* Integrar la funcionalidad con el servicio de correo o de mensajería (según la tecnología definida).
  - *Subtarea 2.4:* Registrar la acción de notificación en el historial de auditoría.
- **Tarea 3: Pruebas y Documentación**
  - *Subtarea 3.1:* Escribir pruebas unitarias e integración (usando Jasmine y escenarios BDD) para el flujo de notificación.
  - *Subtarea 3.2:* Documentar el proceso de notificación en la guía del usuario y la documentación técnica.
  - *Subtarea 3.3:* Incluir la funcionalidad en el pipeline de CI/CD.

RF-21: Generación automática de documentos de aprobación o rechazo, lo que formaliza el proceso y agiliza la comunicación

HU-18: Generación Automática de Documentos de Aprobación o Rechazo

Historia de Usuario

Como **Responsable de Validación**, quiero **que el sistema genere automáticamente un documento PDF de aprobación o rechazo de reserva** para formalizar y documentar de manera oficial la decisión tomada, facilitando la comunicación con el solicitante y garantizando la trazabilidad del proceso.

Criterios de Aceptación

- Al tomar una decisión (aprobación o rechazo) sobre una solicitud de reserva, se debe generar un documento PDF que incluya:
    - Datos del solicitante (nombre, rol, contacto).
    - Detalles completos de la reserva (recurso, fecha, hora, duración).
    - Estado de la solicitud (aprobada o rechazada).
    - En caso de rechazo, debe incluir el motivo detallado.
    - Información del responsable (nombre, cargo, firma digital o datos de validación) si aplica.
- El documento debe generarse automáticamente al finalizar la acción de validación.
- El PDF debe poder descargarse desde el portal y enviarse automáticamente por correo electrónico al solicitante.
- La generación del documento y el envío deben registrarse en el historial de auditoría.
- Se debe validar que todos los campos obligatorios estén presentes antes de generar el documento.
- La generación y envío no deben afectar significativamente el rendimiento del sistema (tiempo de generación y envío < 2 segundos en condiciones normales).

Para estructurar la solución, se desglosa en dos sub-historias:

SubHU-18.1: Generación Automática del Documento PDF

Historia de Usuario

Como **Responsable de Validación**, quiero **que el sistema genere automáticamente un documento PDF con todos los detalles de la reserva y la decisión** para formalizar de manera oficial la aprobación o rechazo y mantener una documentación confiable del proceso.

Criterios de Aceptación

- Al finalizar la validación de una solicitud, se debe invocar el módulo de generación de PDF.
- El documento debe incluir:
    - Datos del solicitante.
    - Información del recurso (nombre, ubicación, etc.).
    - Fecha, hora y duración de la reserva.
    - Estado de la solicitud y, en caso de rechazo, el motivo.
    - Datos del responsable de la validación.
- El documento se genera en formato PDF y se guarda en un repositorio seguro.
- Se debe registrar la generación del documento en el historial de auditoría (usuario, fecha, acción).
- El proceso de generación debe manejar errores y notificar en caso de fallo.

Tareas y Subtareas

- **Tarea 1: Diseño de la Plantilla del Documento PDF**

    - *Subtarea 1.1:* Definir el contenido y formato de la plantilla (estructura, tipografía, logos institucionales, etc.).
    - *Subtarea 1.2:* Crear mockups y validarlos con stakeholders (Responsables y Administradores).
- **Tarea 2: Desarrollo del Módulo de Generación de PDF en NestJS**
    - *Subtarea 2.1:* Seleccionar una librería adecuada (por ejemplo, PDFKit o Puppeteer).
    - *Subtarea 2.2:* Definir el DTO que recoja todos los datos necesarios de la reserva y la decisión.
    - *Subtarea 2.3:* Implementar la lógica para rellenar la plantilla con los datos dinámicos.
    - *Subtarea 2.4:* Implementar validaciones para asegurar que todos los campos obligatorios estén presentes.
    - *Subtarea 2.5:* Guardar el PDF generado en un sistema de almacenamiento seguro (puede ser local o en la nube).
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
    - *Subtarea 3.1:* Configurar el logging (por ejemplo, con Winston) para registrar la generación del documento.
    - *Subtarea 3.2:* Implementar manejo de excepciones para capturar y notificar errores en el proceso de generación.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración en Jasmine utilizando escenarios Given-When-Then.
    - *Subtarea 4.2:* Documentar la API y la funcionalidad de generación de PDF en la guía del usuario.
    - *Subtarea 4.3:* Incluir el módulo en el pipeline de CI/CD.

SubHU-18.2: Envío Automático y Registro del Documento

Historia de Usuario

Como **Responsable de Validación**, quiero **que el sistema envíe automáticamente el documento PDF generado al solicitante y registre la acción de envío** para garantizar que el usuario reciba la notificación formal de la decisión y se mantenga la trazabilidad de la comunicación.

Criterios de Aceptación

- Una vez generado el documento, el sistema debe enviar automáticamente un correo electrónico (o notificación vía app/WhatsApp) al solicitante con el PDF adjunto.
- El correo/notificación debe incluir un mensaje claro con la decisión (aprobación o rechazo) y, en caso de rechazo, el motivo.
- El documento debe estar disponible para descarga en el perfil del usuario.
- La acción de envío debe quedar registrada en el historial de auditoría (incluyendo detalles del correo, usuario notificado y fecha/hora).
- El proceso de envío debe manejar errores y notificar a un administrador en caso de fallo.

Tareas y Subtareas

- **Tarea 1: Desarrollo del Módulo de Envío de Notificaciones**
    - *Subtarea 1.1:* Definir el DTO para la notificación, incluyendo el documento PDF y los datos relevantes (estado, motivo, etc.).
    - *Subtarea 1.2:* Integrar el módulo con un servicio de correo electrónico o notificaciones (por ejemplo, SendGrid, Amazon SES o similar).
    - *Subtarea 1.3:* Configurar el formato del mensaje y la plantilla de notificación.
- **Tarea 2: Integración con el Módulo de Generación de PDF**
    - *Subtarea 2.1:* Asegurar que el PDF generado se pase correctamente al módulo de notificación.
    - *Subtarea 2.2:* Implementar la lógica para adjuntar el documento y enviar el mensaje al solicitante.
- **Tarea 3: Registro de Auditoría del Envío**
    - *Subtarea 3.1:* Configurar logging para registrar la acción de envío de notificaciones.
    - *Subtarea 3.2:* Implementar el registro en la base de datos de la acción (usuario, fecha, resultado del envío).
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración (BDD con Jasmine y escenarios Given-When-Then) para validar el envío automático.
    - *Subtarea 4.2:* Documentar el proceso de notificación y registro en la guía del usuario y la documentación técnica.
    - *Subtarea 4.3:* Incluir esta funcionalidad en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

RF-22: Notificación automática al solicitante con el estado de la solicitud, crítica para la transparencia

HU-19: Notificación Automática al Solicitante con Carta de Aceptación o Rechazo

Historia de Usuario

Como **Solicitante**, quiero **recibir una notificación automática que incluya la carta formal (en formato PDF) de aceptación o rechazo de mi solicitud de reserva** para estar informado de manera oportuna y tomar las acciones necesarias.

Criterios de Aceptación

- La notificación se debe enviar automáticamente una vez que el responsable haya tomado una decisión sobre la solicitud.
- El mensaje de notificación debe incluir:
    - Estado final de la solicitud (aprobada o rechazada).
    - Detalles relevantes de la reserva (recurso, fecha, hora y duración).
    - En caso de rechazo, el motivo detallado.
    - La carta formal generada en formato PDF adjunta.
- El proceso de envío debe completarse en menos de 2 segundos bajo condiciones normales de carga.

- La notificación debe registrarse en el historial de auditoría (incluyendo el solicitante, fecha, hora y resultado del envío).
- El solicitante debe poder acceder a la carta desde su perfil en la plataforma si, por algún motivo, no se recibe el correo.

Para estructurar la solución se desglosa en dos sub-historias:

SubHU-19.1: Generación de la Notificación con Carta Adjunta

Historia de Usuario

Como **Responsable de Validación**, quiero **que el sistema genere automáticamente una notificación que incluya la carta formal de aprobación o rechazo en PDF** para formalizar la decisión y mantener la documentación del proceso.

Criterios de Aceptación

- Al finalizar la decisión (aprobación o rechazo), el sistema debe invocar el módulo de generación de notificación.
- La notificación debe incluir un mensaje que resuma la decisión y un enlace o adjunto con la carta PDF.
- La carta PDF debe contener:
    - Datos del solicitante.
    - Detalles de la reserva (recurso, fecha, hora, duración).
    - Estado de la solicitud y, en caso de rechazo, el motivo.
    - Datos del responsable de la validación (nombre, cargo y, si aplica, firma digital).
- Se debe validar que todos los datos obligatorios estén presentes antes de generar la notificación.
- La generación de la notificación debe quedar registrada en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Plantilla de Notificación y Carta**
    - *Subtarea 1.1:* Definir el contenido, formato y estructura de la carta en PDF.
    - *Subtarea 1.2:* Crear mockups de la notificación que incluya el mensaje y el adjunto.
    - *Subtarea 1.3:* Validar el diseño con responsables y solicitantes.
- **Tarea 2: Desarrollo del Módulo de Generación de Notificación en NestJS**
    - *Subtarea 2.1:* Definir el DTO que recopile la información necesaria (datos del solicitante, detalles de reserva, estado, motivo, datos del responsable).
    - *Subtarea 2.2:* Implementar la lógica para rellenar la plantilla de la carta y generar el PDF usando una librería (por ejemplo, PDFKit o Puppeteer).
    - *Subtarea 2.3:* Integrar la generación del PDF con el módulo de notificación.
    - *Subtarea 2.4:* Validar la integridad de los datos antes de la generación.
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
    - *Subtarea 3.1:* Configurar logging para registrar la generación de la notificación y el PDF.

    - *Subtarea 3.2:* Implementar manejo de errores para notificar a los administradores en caso de fallo en la generación.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Escribir pruebas unitarias e integración en Jasmine utilizando escenarios Given-When-Then.
    - *Subtarea 4.2:* Documentar la funcionalidad en la guía del usuario y la documentación técnica.
    - *Subtarea 4.3:* Integrar la funcionalidad en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

SubHU-19.2: Envío Automático y Registro de la Notificación

Historia de Usuario

Como **Responsable de Validación**, quiero **que el sistema envíe automáticamente la notificación con la carta adjunta al solicitante y registre la acción** para asegurar que el usuario reciba la comunicación formal de la decisión y se mantenga la trazabilidad del proceso.

Criterios de Aceptación

- Una vez generada la notificación, el sistema debe enviar automáticamente un correo electrónico (u otra modalidad de notificación configurada) al solicitante con el PDF adjunto.
- El mensaje de notificación debe incluir un resumen claro de la decisión (aprobación o rechazo) y, en caso de rechazo, el motivo.
- El envío debe completarse en menos de 2 segundos en condiciones normales.
- La notificación debe quedar accesible desde el perfil del solicitante.
- La acción de envío y cualquier error ocurrido deben registrarse en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Desarrollo del Módulo de Envío de Notificaciones**
    - *Subtarea 1.1:* Definir el DTO para el envío de notificaciones, incluyendo datos de la notificación y el PDF.
    - *Subtarea 1.2:* Integrar el módulo con un servicio de correo electrónico (por ejemplo, SendGrid, Amazon SES) o sistema de mensajería.
    - *Subtarea 1.3:* Configurar el formato y la plantilla del correo electrónico.
- **Tarea 2: Integración con el Módulo de Generación de Notificación**
    - *Subtarea 2.1:* Asegurar que el PDF generado en SubHU-19.1 se adjunte correctamente al correo.
    - *Subtarea 2.2:* Implementar la lógica para enviar la notificación inmediatamente después de la generación.
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones en el Envío**
    - *Subtarea 3.1:* Configurar logging para registrar el envío de notificaciones (usuario, fecha, resultado).
    - *Subtarea 3.2:* Implementar manejo de errores para capturar y notificar fallos en el envío.

- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine con escenarios Given-When-Then para el flujo de envío.
    - *Subtarea 4.2:* Documentar el proceso de notificación en la guía del usuario y en la documentación técnica.
    - *Subtarea 4.3:* Incluir esta funcionalidad en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

Should - Módulo de Aprobaciones y Gestión de Solicitudes

RF-23: Pantalla de control para el personal de vigilancia, permitiendo un seguimiento en tiempo real de las reservas aprobadas

HU-20: Pantalla de Control para el Personal de Vigilancia

Historia de Usuario

Como **Personal de Vigilancia**, quiero **visualizar en tiempo real una pantalla que muestre todas las reservas aprobadas (para el día y próximas jornadas)** para gestionar y controlar el acceso a los recursos, asegurando que solo las personas autorizadas ingresen a los espacios asignados.

Criterios de Aceptación

- La pantalla debe mostrar en tiempo real una lista de reservas aprobadas para el día y las próximas jornadas.
- Cada registro debe incluir:
    - Datos del solicitante (nombre, identificación, rol).
    - Detalles de la reserva (recurso, fecha, hora de inicio y fin, ubicación).
    - Estado de la reserva (confirmada, en uso, cancelada).
- Se debe incluir un buscador y filtros por fecha, recurso, ubicación y nombre/ID del solicitante.
- La pantalla debe permitir acciones para:
    - Marcar manualmente la entrada del usuario (cambio de estado a “En Uso”).
    - Registrar incidencias o irregularidades (con un campo de texto para comentarios).
    - Permitir ver una opción para búsqueda manual en caso de que el usuario no aparezca en la lista.
- Toda acción realizada (confirmación de entrada, registro de incidencia, búsqueda manual) debe quedar registrada en el historial de auditoría.
- La interfaz debe actualizarse automáticamente en tiempo real cuando se produzcan cambios en las reservas (nuevas aprobaciones, cancelaciones, modificaciones).
- La pantalla debe ser responsiva y usable en dispositivos móviles y de escritorio.

Para cubrir integralmente el requerimiento se desglosa en dos sub-historias:

SubHU-20.1: Visualización y Filtrado en Tiempo Real

Historia de Usuario

Como **Personal de Vigilancia**, quiero **visualizar y filtrar en tiempo real las reservas aprobadas** para localizar rápidamente la información necesaria y gestionar el acceso de forma eficiente.

Criterios de Aceptación

- La pantalla actualiza la lista de reservas aprobadas en tiempo real (con un tiempo de respuesta menor a 2 segundos).
- Se deben aplicar filtros por fecha, recurso, ubicación y nombre o identificación del solicitante.
- Los resultados muestran todos los datos relevantes (nombre, identificación, recurso, fecha, hora, ubicación, estado).
- La acción de filtrar debe quedar registrada en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Visualización**
  - *Subtarea 1.1:* Crear wireframes y mockups de la pantalla de control con un listado de reservas, filtros y buscador.
  - *Subtarea 1.2:* Validar el diseño con personal de vigilancia y ajustar según retroalimentación.
- **Tarea 2: Desarrollo del Componente de Visualización (Frontend)**
  - *Subtarea 2.1:* Implementar el componente utilizando una librería adecuada (por ejemplo, una tabla dinámica o grid).
  - *Subtarea 2.2:* Integrar filtros y opciones de búsqueda.
  - *Subtarea 2.3:* Configurar la actualización en tiempo real mediante WebSockets o polling.
- **Tarea 3: Desarrollo del Endpoint de Consulta (Backend)**
  - *Subtarea 3.1:* Definir el DTO para la consulta de reservas aprobadas.
  - *Subtarea 3.2:* Implementar la lógica en el servicio para recuperar reservas y aplicar filtros.
  - *Subtarea 3.3:* Asegurar el manejo de excepciones y validaciones de datos.
- **Tarea 4: Registro de Auditoría y Pruebas**
  - *Subtarea 4.1:* Configurar logging para registrar cada acción de filtrado y actualización.
  - *Subtarea 4.2:* Desarrollar pruebas unitarias e integración con Jasmine (BDD: Given-When-Then).
  - *Subtarea 4.3:* Documentar la funcionalidad y actualizar la guía del usuario.

SubHU-20.2: Confirmación de Entrada y Registro de Incidencias

Historia de Usuario

Como **Personal de Vigilancia**, quiero **marcar la entrada de los usuarios y registrar incidencias directamente desde la pantalla de control** para confirmar el acceso de manera ordenada y documentar cualquier irregularidad.

Criterios de Aceptación

- La pantalla debe incluir opciones para marcar la entrada de un usuario (cambio de estado a “En Uso”) y para registrar incidencias.
- Al confirmar la entrada, el sistema debe actualizar el estado de la reserva y registrar la acción (usuario, fecha, hora) en el historial de auditoría.

- Se debe disponer de un formulario para ingresar incidencias, con un campo de texto para comentarios.
- Si el usuario no aparece en la lista, se debe permitir la validación manual mediante ingreso de identificación.
- La interfaz debe actualizarse en tiempo real tras cada acción.

Tareas y Subtareas

- **Tarea 1: Diseño del Flujo de Confirmación y Registro de Incidencias**
    - *Subtarea 1.1:* Crear wireframes y mockups que ilustren el flujo para confirmar entrada y registrar incidencias.
    - *Subtarea 1.2:* Validar el diseño con personal de vigilancia y ajustar según sus requerimientos.
- **Tarea 2: Desarrollo del Componente de Confirmación en el Frontend**
    - *Subtarea 2.1:* Implementar botones y formularios para marcar la entrada y registrar incidencias.
    - *Subtarea 2.2:* Integrar la función de validación manual para buscar usuario por identificación.
    - *Subtarea 2.3:* Asegurar que la pantalla se actualice en tiempo real con cada acción.
- **Tarea 3: Desarrollo del Endpoint de Actualización de Estado y Registro de Incidencias (Backend)**
    - *Subtarea 3.1:* Definir el DTO para actualizar el estado de la reserva y registrar una incidencia.
    - *Subtarea 3.2:* Implementar la lógica en el servicio para actualizar el estado a “En Uso” y almacenar la incidencia.
    - *Subtarea 3.3:* Integrar validaciones y manejo de excepciones para evitar inconsistencias.
    - *Subtarea 3.4:* Registrar la acción en el historial de auditoría.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración (usando Jasmine y escenarios BDD).
    - *Subtarea 4.2:* Documentar el flujo completo en la guía del usuario y actualizar la documentación técnica.
    - *Subtarea 4.3:* Integrar la funcionalidad en el pipeline de CI/CD.

RF-24: Configuración de flujos de aprobación diferenciados según el tipo de usuario, que ayuda a adaptar el proceso a las necesidades de cada perfil

HU-21: Configuración de Flujos de Aprobación Diferenciados

Historia de Usuario

Como **Administrador**, quiero **configurar diferentes flujos de aprobación según el tipo de usuario (estudiante, profesor, administrativo)** para que cada solicitud de reserva se valide con los niveles y responsables adecuados, optimizando la eficiencia y el cumplimiento de las políticas institucionales.

Criterios de Aceptación

- El sistema debe permitir definir y asignar flujos de aprobación diferenciados para cada tipo de usuario.
- Cada flujo debe incluir:
    - La cantidad de niveles de aprobación (por ejemplo, un estudiante requiere aprobación de un profesor y, en algunos casos, de un director).
    - El rol del aprobador en cada nivel (por ejemplo, profesor, director, ingeniero de soporte).
    - Un tiempo límite configurable para la respuesta de cada aprobador.
- El sistema debe enviar notificaciones automáticas al aprobador asignado cuando se active una solicitud.
- Si el aprobador no responde en el tiempo configurado, el sistema debe permitir la escalada o reasignación de la solicitud a otro responsable.
- El solicitante debe poder consultar el estado de su solicitud en tiempo real.
- Todas las configuraciones, acciones de aprobación, escaladas y notificaciones deben quedar registradas en el historial de auditoría.

Para estructurar la solución se desglosa en dos sub-historias:

SubHU-21.1: Configuración de Flujos de Aprobación por Tipo de Usuario

Historia de Usuario

Como **Administrador**, quiero **configurar de forma diferenciada los flujos de aprobación para estudiantes, profesores y personal administrativo** para que cada solicitud siga un proceso de validación acorde a las políticas y características de cada perfil.

Criterios de Aceptación

- Se debe disponer de una interfaz para definir:
    - El número de niveles de aprobación para cada tipo de usuario.
    - El rol y responsable asignado en cada nivel (por ejemplo, profesor para estudiantes; director para docentes; aprobación directa para administrativos).
    - El tiempo límite para la respuesta en cada nivel.
- La configuración debe permitir editar, activar o desactivar flujos sin afectar solicitudes en curso.
- Los cambios en la configuración deben registrarse en el historial de auditoría (usuario, fecha, modificación).
- La interfaz debe ofrecer opciones para previsualizar y confirmar la configuración antes de guardarla.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Configuración de Flujos**
    - *Subtarea 1.1:* Crear wireframes y mockups de la pantalla de configuración de flujos de aprobación.
    - *Subtarea 1.2:* Incluir secciones diferenciadas para cada tipo de usuario (estudiante, profesor, administrativo).

    - *Subtarea 1.3:* Validar el diseño con stakeholders y ajustar según retroalimentación.
- **Tarea 2: Desarrollo del Endpoint para Configuración de Flujos**
    - *Subtarea 2.1:* Definir el DTO que incluya número de niveles, roles de aprobadores y tiempos límite.
    - *Subtarea 2.2:* Implementar la lógica en el servicio de configuración de aprobaciones.
    - *Subtarea 2.3:* Integrar el endpoint con la base de datos utilizando Prisma y MongoDB.
    - *Subtarea 2.4:* Incluir validaciones (por ejemplo, que los tiempos sean positivos, que se asigne al menos un aprobador por flujo).
- **Tarea 3: Registro de Auditoría y Manejo de Errores**
    - *Subtarea 3.1:* Configurar logging para registrar cada acción de configuración.
    - *Subtarea 3.2:* Implementar manejo de excepciones para capturar errores y notificar al administrador.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración en Jasmine (BDD: Given-When-Then).
    - *Subtarea 4.2:* Documentar el proceso y la interfaz de configuración en la guía del usuario y la documentación técnica.
    - *Subtarea 4.3:* Integrar el módulo en el pipeline de CI/CD.

SubHU-21.2: Notificación y Escalado en Flujos de Aprobación

Historia de Usuario

Como **Administrador**, quiero **configurar notificaciones automáticas y reglas de escalado en los flujos de aprobación** para que, en caso de no respuesta de un aprobador, la solicitud se reasigne o escale automáticamente, asegurando la continuidad del proceso de validación.

Criterios de Aceptación

- El sistema debe enviar notificaciones automáticas al aprobador asignado cuando se active una solicitud.
- Si el aprobador no responde en el tiempo configurado, se debe notificar la necesidad de escalado o reasignación.
- La interfaz para escalado manual (si es necesario) debe permitir al administrador seleccionar otro responsable.
- El solicitante debe ser notificado de cualquier cambio en el flujo de aprobación (por ejemplo, reasignación a otro aprobador).
- Todas las notificaciones y escaladas deben quedar registradas en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño del Módulo de Notificaciones y Escalado**
    - *Subtarea 1.1:* Crear wireframes y mockups de la interfaz de notificaciones para aprobadores y de escalado manual.

    - *Subtarea 1.2:* Validar el diseño con stakeholders (administradores y responsables de validación).
- **Tarea 2: Desarrollo del Módulo de Notificación**
    - *Subtarea 2.1:* Definir el DTO para enviar notificaciones que incluya detalles de la solicitud y el tiempo límite.
    - *Subtarea 2.2:* Integrar el módulo con un servicio de correo electrónico o de notificaciones (por ejemplo, SendGrid).
    - *Subtarea 2.3:* Implementar la lógica para enviar notificaciones al aprobador al iniciar la solicitud.
- **Tarea 3: Desarrollo de la Lógica de Escalado Automático**
    - *Subtarea 3.1:* Implementar en el servicio de reservas la lógica para detectar falta de respuesta en el tiempo configurado.
    - *Subtarea 3.2:* Configurar la reasignación automática o escalado a un nivel superior.
    - *Subtarea 3.3:* Notificar al solicitante sobre el cambio en el flujo de aprobación.
- **Tarea 4: Registro de Auditoría y Manejo de Excepciones**
    - *Subtarea 4.1:* Configurar logging para registrar cada notificación y acción de escalado.
    - *Subtarea 4.2:* Implementar manejo de excepciones y notificación de fallos en el proceso.
- **Tarea 5: Pruebas y Documentación**
    - *Subtarea 5.1:* Escribir pruebas unitarias e integración en Jasmine con escenarios Given-When-Then.
    - *Subtarea 5.2:* Documentar el proceso de notificación y escalado en la guía del usuario y la documentación técnica.
    - *Subtarea 5.3:* Integrar la funcionalidad en el pipeline de CI/CD.

RF-25: Registro y trazabilidad de todas las aprobaciones, para auditoría y control

HU-22: Registro y Trazabilidad de Aprobaciones para Auditoría

Historia de Usuario

Como **Administrador**, quiero **que el sistema registre y mantenga un historial detallado de todas las aprobaciones, modificaciones y rechazos de solicitudes de reserva** para auditar el proceso, garantizar transparencia y tomar decisiones informadas basadas en datos históricos.

Criterios de Aceptación

- Cada acción (aprobación, modificación o rechazo) debe registrarse automáticamente con los siguientes datos:
    - Identificación del solicitante y del recurso (nombre, ubicación, fecha, hora, duración).
    - Estado de la solicitud (pendiente, aprobada, rechazada, modificada).
    - Datos del aprobador (nombre, cargo, comentarios de la acción).

    - Fecha y hora exacta de la acción.
- Los registros deben ser inalterables y accesibles solo para usuarios autorizados.
- La interfaz de consulta debe permitir filtrar por fecha, usuario, recurso, estado y aprobador.
- Se debe ofrecer la opción de exportar el historial filtrado en formato CSV.
- El tiempo de respuesta de las consultas debe ser menor a 2 segundos en condiciones normales.
- Toda acción de registro debe quedar documentada en el historial de auditoría.

Para cubrir integralmente el requerimiento se desglosa en dos sub-historias:

SubHU-22.1: Registro Automático de Acciones de Aprobación

Historia de Usuario

Como **Administrador**, quiero **que el sistema registre automáticamente cada acción de validación (aprobación, rechazo o solicitud de modificación) de una solicitud de reserva** para disponer de un historial completo y verificable para auditorías internas y externas.

Criterios de Aceptación

- Al tomar una decisión sobre una solicitud, el sistema debe generar un registro que incluya todos los datos obligatorios:
    - Datos del solicitante, detalles de la reserva, estado final, datos del aprobador, fecha y hora, y comentarios (en caso de rechazo o modificación).
- El registro debe asignarse un identificador único y almacenarse de forma inalterable.
- Se debe notificar al responsable de la acción (por ejemplo, mediante un mensaje en la interfaz) que el registro se ha guardado exitosamente.
- En caso de error en el registro, se debe notificar al administrador y registrar el fallo.

Tareas y Subtareas

- **Tarea 1: Diseño del Modelo de Datos del Historial de Aprobaciones**
    - *Subtarea 1.1:* Definir el esquema del historial (campos, tipos de datos, identificador único).
    - *Subtarea 1.2:* Validar el diseño con el equipo de auditoría y seguridad.
- **Tarea 2: Desarrollo del Módulo de Registro Automático en NestJS**
    - *Subtarea 2.1:* Implementar un interceptor o middleware en el servicio de validación que capture cada acción.
    - *Subtarea 2.2:* Desarrollar la lógica en el servicio para almacenar los registros en la base de datos (usando Prisma y MongoDB).
    - *Subtarea 2.3:* Incluir validaciones para asegurar que cada registro contenga todos los datos requeridos.
- **Tarea 3: Configuración de Auditoría y Manejo de Excepciones**
    - *Subtarea 3.1:* Configurar herramientas de logging (por ejemplo, Winston) para registrar cada acción.
    - *Subtarea 3.2:* Implementar manejo de errores para capturar y notificar fallos en el registro.

- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración en Jasmine utilizando escenarios BDD (Given-When-Then).
    - *Subtarea 4.2:* Documentar el modelo de datos, el flujo de registro y las políticas de auditoría en la guía del usuario.
    - *Subtarea 4.3:* Integrar el módulo en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

SubHU-22.2: Consulta y Exportación del Historial de Aprobaciones

Historia de Usuario

Como **Administrador**, quiero **consultar y exportar el historial de aprobaciones de solicitudes de reserva** para auditar el proceso, identificar patrones y mejorar la gestión de las reservas mediante análisis de datos.

Criterios de Aceptación

- La interfaz debe permitir visualizar una lista completa del historial con todos los datos registrados (solicitante, reserva, aprobador, fecha, hora, estado y comentarios).
- Debe incluir filtros por fecha, usuario, recurso, estado y aprobador.
- La consulta debe responder en menos de 2 segundos en condiciones normales.
- Se debe permitir exportar el historial filtrado en formato CSV.
- El acceso a esta funcionalidad debe estar restringido a usuarios autorizados.
- Cada consulta y exportación debe quedar registrada en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Consulta del Historial**
    - *Subtarea 1.1:* Crear wireframes y mockups de la pantalla de consulta, incluyendo filtros y opción de exportación.
    - *Subtarea 1.2:* Validar el diseño con administradores y el equipo de auditoría.
- **Tarea 2: Desarrollo del Endpoint de Consulta en NestJS**
    - *Subtarea 2.1:* Definir el DTO para la consulta con filtros (fecha, usuario, recurso, estado, aprobador).
    - *Subtarea 2.2:* Implementar la lógica en el servicio para recuperar y formatear los registros del historial.
    - *Subtarea 2.3:* Optimizar la consulta para garantizar un tiempo de respuesta inferior a 2 segundos.
- **Tarea 3: Desarrollo de la Funcionalidad de Exportación a CSV**
    - *Subtarea 3.1:* Implementar la generación de un archivo CSV a partir de los registros filtrados.
    - *Subtarea 3.2:* Integrar la opción de exportación en la interfaz de usuario.
- **Tarea 4: Registro de Auditoría y Manejo de Errores**
    - *Subtarea 4.1:* Configurar logging para registrar cada consulta y exportación.
    - *Subtarea 4.2:* Implementar manejo de excepciones para capturar y notificar errores en la consulta o exportación.
- **Tarea 5: Pruebas y Documentación**

- *Subtarea 5.1:* Escribir pruebas unitarias e integración utilizando Jasmine y escenarios BDD (Given-When-Then).
- *Subtarea 5.2:* Documentar el proceso de consulta y exportación en la guía del usuario y la documentación técnica.
- *Subtarea 5.3:* Integrar la funcionalidad en el pipeline de CI/CD.

Could - Módulo de Aprobaciones y Gestión de Solicitudes

RF-26: Implementación de check-in/check-out digital, que podría añadirse en una iteración posterior para validar el uso real del recurso

HU-23: Implementación de Check-In/Check-Out Digital

Historia de Usuario

Como **Usuario que tiene una reserva**, quiero **realizar check-in y check-out digital** para confirmar mi presencia al iniciar y finalizar el uso del recurso, de modo que se optimice la asignación, se libere el recurso oportunamente y se mantenga la trazabilidad del uso.

Criterios de Aceptación Generales

- Se debe ofrecer la opción de realizar check-in digital antes del inicio de la reserva y check-out al finalizar el uso.
- La verificación de identidad se realizará mediante un método seguro (por ejemplo, ingreso de PIN, credenciales o confirmación en la app).
- El sistema registrará la hora exacta del check-in y del check-out.
- Si el usuario no realiza el check-in dentro de un tiempo límite configurado (por ejemplo, 15 minutos tras el inicio de la reserva), se debe notificar a los administradores.
- En caso de check-out anticipado, el recurso se liberará para ser reservado nuevamente.
- Todas las acciones (check-in, check-out, notificaciones) se deben registrar en el historial de auditoría.
- La funcionalidad debe integrarse de forma transparente con el módulo de reservas y actualizar el estado de la reserva en tiempo real.

Para una implementación integral, se desglosa en dos sub-historias:

SubHU-23.1: Check-In Digital

Historia de Usuario

Como **Usuario con reserva activa**, quiero **realizar el check-in digital** mediante un método seguro para confirmar mi presencia y activar el uso del recurso asignado.

Criterios de Aceptación

- La interfaz debe mostrar la opción de check-in digital al iniciar la reserva.
- El usuario debe autenticarse mediante un método seguro (PIN, credenciales o notificación en la app).
- Al realizar el check-in, se debe registrar la hora exacta y actualizar el estado de la reserva a “En Uso”.

- Si el usuario no realiza el check-in dentro del plazo configurado (ej., 15 minutos después del inicio programado), se debe enviar una notificación automática a los administradores.
- La acción de check-in debe quedar registrada en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Check-In**
  - *Subtarea 1.1:* Crear wireframes y mockups para la pantalla de check-in, mostrando opción de autenticación (campo de PIN, botón de confirmación, etc.).
  - *Subtarea 1.2:* Validar el diseño con usuarios y personal de soporte.
- **Tarea 2: Desarrollo del Endpoint de Check-In en NestJS**
  - *Subtarea 2.1:* Definir el DTO que reciba la solicitud de check-in (ID de la reserva, método de autenticación, PIN/credenciales).
  - *Subtarea 2.2:* Implementar la lógica en el servicio para validar la identidad, registrar la hora exacta de check-in y actualizar el estado de la reserva a “En Uso”.
  - *Subtarea 2.3:* Integrar el endpoint con la base de datos utilizando Prisma y MongoDB.
  - *Subtarea 2.4:* Incluir validaciones (por ejemplo, verificación de PIN correcto y del tiempo límite).
- **Tarea 3: Implementación de Notificaciones y Registro de Auditoría**
  - *Subtarea 3.1:* Configurar el envío de notificaciones automáticas al administrador si no se realiza el check-in en el tiempo configurado.
  - *Subtarea 3.2:* Registrar la acción de check-in (usuario, fecha, hora, estado actualizado) en el historial de auditoría utilizando un sistema de logging (por ejemplo, Winston).
- **Tarea 4: Pruebas y Documentación**
  - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine y escenarios BDD (Given-When-Then) para el flujo de check-in.
  - *Subtarea 4.2:* Documentar la funcionalidad de check-in en la guía del usuario y en la documentación técnica.
  - *Subtarea 4.3:* Integrar el módulo de check-in en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

SubHU-23.2: Check-Out Digital

Historia de Usuario

Como **Usuario que está utilizando un recurso**, quiero **realizar el check-out digital** para confirmar el fin de mi uso y liberar el recurso de forma anticipada si es necesario, optimizando la disponibilidad.

Criterios de Aceptación

- La interfaz debe mostrar la opción de check-out digital al finalizar el uso del recurso.
- Al realizar el check-out, se debe registrar la hora exacta y actualizar el estado de la reserva a “Finalizada”.

- Si el usuario realiza el check-out antes del tiempo programado, el recurso se libera inmediatamente para nuevas reservas.
- La acción de check-out debe quedar registrada en el historial de auditoría.
- En caso de no realizar el check-out de forma manual, el sistema debe registrar el cierre automático al final del tiempo programado y notificar al usuario.
- Se debe enviar una notificación de confirmación al usuario y actualizar la disponibilidad en tiempo real.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Check-Out**
    - *Subtarea 1.1:* Crear wireframes y mockups de la pantalla de check-out, mostrando botón de confirmación y estado de la reserva.
    - *Subtarea 1.2:* Validar el diseño con usuarios y personal de soporte, asegurando facilidad de uso en dispositivos móviles y de escritorio.
- **Tarea 2: Desarrollo del Endpoint de Check-Out en NestJS**
    - *Subtarea 2.1:* Definir el DTO para la solicitud de check-out (ID de reserva, datos de autenticación si aplica).
    - *Subtarea 2.2:* Implementar la lógica en el servicio para registrar la hora exacta de check-out, actualizar el estado de la reserva a “Finalizada” y liberar el recurso.
    - *Subtarea 2.3:* Integrar el endpoint con la base de datos usando Prisma y MongoDB.
    - *Subtarea 2.4:* Incluir validaciones y manejo de excepciones para asegurar la correcta actualización del estado.
- **Tarea 3: Notificaciones y Registro de Auditoría**
    - *Subtarea 3.1:* Configurar el envío de notificaciones al usuario confirmando el check-out y la liberación del recurso.
    - *Subtarea 3.2:* Registrar la acción de check-out en el historial de auditoría, incluyendo datos relevantes (usuario, fecha, hora, estado).
    - *Subtarea 3.3:* Implementar lógica de cierre automático en caso de no check-out manual.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración con Jasmine, utilizando escenarios BDD (Given-When-Then) para el flujo de check-out.
    - *Subtarea 4.2:* Documentar la funcionalidad de check-out en la guía del usuario y la documentación técnica.
    - *Subtarea 4.3:* Integrar el módulo de check-out en el pipeline de CI/CD.

RF-27/RF-28: Integración con sistemas de mensajería para notificaciones adicionales (correo, WhatsApp) como mejora complementaria

HU-24: Integración con Sistemas de Mensajería para Notificar Cambios o Cancelaciones

Historia de Usuario

Como **Administrador**, quiero **integrar el sistema con plataformas de mensajería (correo electrónico y WhatsApp)** para notificar automáticamente a los usuarios sobre cambios o cancelaciones en sus reservas y así mejorar la comunicación y reducir la carga de seguimiento manual.

Criterios de Aceptación

- El sistema debe ofrecer una interfaz de configuración donde el administrador pueda definir los canales de mensajería (correo y/o WhatsApp) y las credenciales necesarias.
- Al producirse un cambio o cancelación en una reserva, el sistema debe enviar automáticamente una notificación que incluya:
    - Nombre del usuario.
    - Fecha, hora y detalles de la reserva.
    - Motivo del cambio o cancelación.
- El proceso de notificación debe completarse en menos de 2 segundos bajo condiciones normales de carga.
- Se debe manejar el error en el envío y notificar al administrador en caso de fallo.
- Todas las notificaciones enviadas deben quedar registradas en el historial de auditoría.

SubHU-24.1: Configuración de Mensajería

Historia de Usuario

Como **Administrador**, quiero **configurar los canales de mensajería y credenciales de integración (correo y WhatsApp)** para que el sistema pueda enviar notificaciones de forma segura y personalizada.

Criterios de Aceptación

- La interfaz debe permitir ingresar y guardar las credenciales necesarias (API Keys, Client IDs, URLs, etc.) para correo y WhatsApp.
- Se debe validar la conexión mediante pruebas de autenticación.
- La configuración debe ser editable y guardarse de forma segura.
- Los cambios en la configuración deben registrarse en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Configuración de Mensajería**
    - *Subtarea 1.1:* Crear wireframes y mockups de la pantalla de configuración que incluya campos para cada canal.
    - *Subtarea 1.2:* Validar el diseño con stakeholders y ajustar según retroalimentación.
- **Tarea 2: Desarrollo del Endpoint de Configuración**
    - *Subtarea 2.1:* Definir el DTO para la configuración de mensajería.

    - *Subtarea 2.2:* Implementar la lógica en el servicio para guardar y actualizar las credenciales.
    - *Subtarea 2.3:* Realizar una prueba de conexión para cada canal (correo y WhatsApp).
    - *Subtarea 2.4:* Integrar el endpoint con la base de datos utilizando Prisma y MongoDB.
- **Tarea 3: Registro de Auditoría y Manejo de Errores**
    - *Subtarea 3.1:* Configurar logging para registrar cada cambio en la configuración.
    - *Subtarea 3.2:* Implementar manejo de excepciones para notificar errores en la conexión.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración con Jasmine (Given-When-Then).
    - *Subtarea 4.2:* Documentar la configuración de mensajería en la guía del usuario.
    - *Subtarea 4.3:* Incluir esta funcionalidad en el pipeline de CI/CD.

SubHU-24.2: Envío Automático de Notificaciones por Mensajería

Historia de Usuario

Como **Administrador**, quiero **que el sistema envíe notificaciones automáticas por correo y WhatsApp ante cambios o cancelaciones en reservas** para que los usuarios sean informados de forma inmediata y se reduzca la incertidumbre sobre el estado de sus reservas.

Criterios de Aceptación

- Al producirse un cambio o cancelación en una reserva, el sistema debe enviar una notificación automática que incluya:
    - Información de la reserva (recurso, fecha, hora, duración).
    - Motivo del cambio o cancelación.
    - Instrucciones o recomendaciones en caso de ser necesario.
- La notificación debe enviarse utilizando los canales configurados (correo y/o WhatsApp) y completarse en menos de 2 segundos.
- El proceso de envío debe manejar errores y notificar al administrador en caso de fallo.
- La acción de envío debe quedar registrada en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Plantilla de Notificación**
    - *Subtarea 1.1:* Crear mockups de la notificación, definiendo el contenido y formato (texto, diseño, adjuntos si aplica).
    - *Subtarea 1.2:* Validar la plantilla con stakeholders.
- **Tarea 2: Desarrollo del Módulo de Envío Automático**
    - *Subtarea 2.1:* Definir el DTO para el envío de notificaciones (incluyendo datos de la reserva y motivo).

    - *Subtarea 2.2:* Implementar la lógica en el servicio para enviar notificaciones a través de las APIs configuradas.
    - *Subtarea 2.3:* Integrar validaciones para verificar que los canales de mensajería estén configurados correctamente.
    - *Subtarea 2.4:* Manejar excepciones y errores en el envío, notificando al administrador si es necesario.
- **Tarea 3: Registro de Auditoría y Pruebas**
    - *Subtarea 3.1:* Configurar logging para registrar cada notificación enviada.
    - *Subtarea 3.2:* Desarrollar pruebas unitarias e integración con Jasmine utilizando escenarios Given-When-Then.
    - *Subtarea 3.3:* Documentar el proceso de envío y las configuraciones de notificación.
- **Tarea 4: Integración y Documentación**
    - *Subtarea 4.1:* Integrar el módulo de notificación en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).
    - *Subtarea 4.2:* Actualizar la documentación técnica y la guía del usuario con ejemplos de uso.

HU-25: Notificaciones Automáticas sobre Confirmación, Cancelación o Modificación de Reservas vía Email o WhatsApp

Historia de Usuario

Como **Usuario**, quiero **recibir notificaciones automáticas vía email o WhatsApp sobre la confirmación, cancelación o modificación de mis reservas** para estar informado de forma inmediata y tomar las acciones correspondientes sin tener que consultar manualmente el sistema.

Criterios de Aceptación

- El sistema debe enviar notificaciones automáticas por correo y/o WhatsApp en los siguientes eventos:
    - Confirmación de reserva.
    - Cancelación de reserva.
    - Modificación de reserva.
- Las notificaciones deben incluir:
    - Detalles de la reserva (recurso, fecha, hora, duración).
    - Estado actualizado (confirmada, cancelada, modificada).
    - En caso de modificación o cancelación, motivo o comentarios relevantes.
- El envío debe completarse en menos de 2 segundos bajo condiciones normales.
- Las notificaciones deben quedar registradas en el historial de auditoría.
- El usuario debe poder configurar su método de notificación preferido desde su perfil.
- Se debe notificar al usuario mediante los canales configurados y, en caso de error, registrar la incidencia y notificar al administrador.

Tareas y Subtareas

- **Tarea 1: Diseño de la Plantilla de Notificación Automática**

    - *Subtarea 1.1:* Crear wireframes y mockups de la plantilla de notificación para confirmación, cancelación y modificación.
    - *Subtarea 1.2:* Validar el diseño con usuarios y administradores, ajustando el contenido y formato.
- **Tarea 2: Desarrollo del Endpoint para Notificaciones Automáticas**
    - *Subtarea 2.1:* Definir el DTO para el envío de notificaciones, incluyendo todos los datos relevantes.
    - *Subtarea 2.2:* Implementar la lógica en el servicio de reservas para disparar notificaciones en cada evento (confirmación, cancelación, modificación).
    - *Subtarea 2.3:* Integrar la lógica con el módulo de mensajería configurado previamente (HU-24).
    - *Subtarea 2.4:* Incluir validaciones para verificar que el usuario tenga configurado su método de notificación preferido.
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
    - *Subtarea 3.1:* Configurar el logging para registrar cada notificación enviada (usuario, fecha, tipo de evento).
    - *Subtarea 3.2:* Implementar manejo de errores que capture fallos en el envío y notifique al administrador.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración en Jasmine utilizando escenarios Given-When-Then.
    - *Subtarea 4.2:* Documentar la funcionalidad y actualizar la guía del usuario.
    - *Subtarea 4.3:* Incluir la funcionalidad en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

Must - Módulo de Reportes y Análisis

RF-31: Generación de reportes sobre la utilización de recursos (por programa académico, período y tipo de recurso), crucial para la toma de decisiones

HU-26: Generación de Reportes sobre la Utilización de Recursos

Historia de Usuario

Como **Administrador**, quiero **generar reportes que muestren la utilización de recursos filtrados por programa académico, período de tiempo y tipo de recurso** para analizar el uso de los espacios y tomar decisiones informadas que optimicen la asignación de recursos.

Criterios de Aceptación

- Se debe permitir aplicar filtros sobre:
    - Programa académico.
    - Período de tiempo (rango de fechas).
    - Tipo de recurso (salón, laboratorio, auditorio, equipo, etc.).
- El reporte debe incluir, al menos, los siguientes datos:
    - Número total de reservas.
    - Promedio de utilización.
    - Recursos con mayor y menor demanda.
    - Porcentaje de uso por programa académico.
- La generación del reporte debe completarse en menos de 2 segundos en condiciones normales.
- El reporte debe visualizarse de forma interactiva (tablas y gráficos) y permitir la actualización dinámica al modificar filtros.
- Se debe permitir exportar el reporte en formato CSV.
- Toda acción de generación y exportación del reporte debe quedar registrada en el historial de auditoría.

Para cubrir integralmente este requerimiento se desglosa en dos sub-historias:

SubHU-26.1: Reporte Interactivo de Utilización de Recursos

Historia de Usuario

Como **Administrador**, quiero **visualizar un reporte interactivo en la plataforma que muestre estadísticas de utilización de recursos según programa, período y tipo** para identificar patrones de uso y optimizar la planificación y asignación de recursos.

Criterios de Aceptación

- La interfaz debe permitir aplicar filtros por programa académico, período y tipo de recurso.

- El reporte interactivo debe mostrar gráficos (barras, líneas) y tablas con los datos relevantes.
- La actualización de los datos al aplicar o modificar filtros debe realizarse en tiempo real (o mediante acción de “Buscar”) en menos de 2 segundos.
- La visualización debe ser responsiva y fácil de interpretar.
- La generación del reporte debe quedar registrada en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Reporte Interactivo**
    - *Subtarea 1.1:* Crear wireframes y mockups del formulario de búsqueda y visualización interactiva.
    - *Subtarea 1.2:* Definir la disposición de filtros y la representación gráfica de los datos.
    - *Subtarea 1.3:* Validar el diseño con administradores y usuarios clave.
- **Tarea 2: Desarrollo del Endpoint de Consulta de Reporte**
    - *Subtarea 2.1:* Definir el DTO que reciba los filtros (programa, período, tipo de recurso).
    - *Subtarea 2.2:* Implementar la lógica en el servicio para recuperar y procesar los datos de reservas.
    - *Subtarea 2.3:* Optimizar la consulta para asegurar un tiempo de respuesta inferior a 2 segundos.
    - *Subtarea 2.4:* Integrar el endpoint con la base de datos usando Prisma y MongoDB.
- **Tarea 3: Desarrollo del Componente de Visualización Interactiva**
    - *Subtarea 3.1:* Seleccionar o desarrollar componentes gráficos (por ejemplo, usando Chart.js o D3.js).
    - *Subtarea 3.2:* Integrar los componentes gráficos con el endpoint de consulta.
    - *Subtarea 3.3:* Implementar la actualización dinámica de datos al modificar los filtros.
- **Tarea 4: Registro de Auditoría y Manejo de Excepciones**
    - *Subtarea 4.1:* Configurar logging para registrar cada generación y actualización del reporte.
    - *Subtarea 4.2:* Implementar manejo de excepciones para capturar y notificar errores en la generación del reporte.
- **Tarea 5: Pruebas y Documentación**
    - *Subtarea 5.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine y escenarios BDD (Given-When-Then).
    - *Subtarea 5.2:* Documentar la funcionalidad en la guía del usuario y la documentación técnica.
    - *Subtarea 5.3:* Integrar la funcionalidad en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

SubHU-26.2: Exportación de Reporte a CSV

Historia de Usuario

Como **Administrador**, quiero **exportar el reporte generado en formato CSV** para poder compartir y analizar los datos en herramientas externas y facilitar la toma de decisiones.

Criterios de Aceptación

- El sistema debe ofrecer un botón o acción de “Exportar a CSV” en la vista del reporte interactivo.
- El archivo CSV debe incluir todos los datos mostrados en el reporte (número total de reservas, promedio de utilización, etc.) y respetar el formato correcto.
- La exportación debe completarse en menos de 2 segundos en condiciones normales.
- La acción de exportación debe quedar registrada en el historial de auditoría.
- El usuario debe poder descargar el archivo CSV y, si es necesario, reintentar la exportación en caso de error.

Tareas y Subtareas

- **Tarea 1: Desarrollo del Endpoint de Exportación**
  - *Subtarea 1.1:* Definir el DTO para la exportación que incluya los filtros aplicados.
  - *Subtarea 1.2:* Implementar la lógica en el servicio para generar el archivo CSV a partir de los datos del reporte.
  - *Subtarea 1.3:* Validar la integridad y formato del CSV generado.
- **Tarea 2: Integración en la Interfaz de Usuario**
  - *Subtarea 2.1:* Añadir un botón “Exportar a CSV” en la vista del reporte interactivo.
  - *Subtarea 2.2:* Conectar el botón con el endpoint de exportación para iniciar la descarga.
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
  - *Subtarea 3.1:* Configurar logging para registrar la acción de exportación.
  - *Subtarea 3.2:* Implementar manejo de excepciones para capturar errores durante la generación o descarga del CSV.
- **Tarea 4: Pruebas y Documentación**
  - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine y escenarios BDD (Given-When-Then) para la exportación.
  - *Subtarea 4.2:* Documentar la funcionalidad de exportación en la guía del usuario y la documentación técnica.
  - *Subtarea 4.3:* Integrar la funcionalidad en el pipeline de CI/CD.

RF-32: Reportes de cantidad de reservas realizadas por usuario o profesor, para monitorear la actividad

HU-27: Generación de Reportes de Reservas por Usuario o Profesor

Historia de Usuario

Como **Administrador**, quiero **generar reportes que muestren la cantidad de reservas realizadas por cada usuario o profesor** para monitorear la actividad, detectar patrones de uso y optimizar la asignación de recursos.

Criterios de Aceptación

- El sistema debe permitir aplicar filtros por:
    - Usuario o profesor.
    - Período de tiempo (rango de fechas).
    - Tipo de recurso (opcional).
- El reporte debe mostrar:
    - Número total de reservas realizadas.
    - Comparativa entre reservas confirmadas y canceladas.
    - Recursos más y menos reservados por cada usuario.
- La generación del reporte debe completarse en menos de 2 segundos en condiciones normales.
- La visualización debe ser interactiva (tablas y gráficos) y responsiva.
- Se debe permitir exportar el reporte en formato CSV.
- Todas las acciones de generación y exportación del reporte deben quedar registradas en el historial de auditoría.

Para cubrir integralmente este requerimiento se desglosa en dos sub-historias:

SubHU-27.1: Reporte Interactivo de Reservas por Usuario o Profesor

Historia de Usuario

Como **Administrador**, quiero **visualizar un reporte interactivo que detalle la cantidad de reservas realizadas por cada usuario o profesor** para identificar tendencias y tomar decisiones informadas sobre la asignación de recursos.

Criterios de Aceptación

- La interfaz debe permitir aplicar filtros por usuario/profesor, período de tiempo y, opcionalmente, tipo de recurso.
- El reporte interactivo debe mostrar:
    - Número total de reservas por usuario.
    - Porcentaje de reservas confirmadas y canceladas.
    - Gráficos y tablas que ilustren la distribución de reservas.
- Los resultados deben actualizarse en tiempo real o mediante acción de “Buscar” con un tiempo de respuesta inferior a 2 segundos.
- La acción de generación del reporte debe quedar registrada en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Reporte Interactivo**

    - *Subtarea 1.1:* Crear wireframes y mockups de la pantalla de reporte interactivo, incluyendo filtros y representación gráfica (por ejemplo, gráficos de barras o líneas).
    - *Subtarea 1.2:* Validar el diseño con administradores y ajustar según feedback.
- **Tarea 2: Desarrollo del Endpoint de Consulta de Reporte**
    - *Subtarea 2.1:* Definir el DTO que reciba los filtros (usuario/profesor, rango de fechas, tipo de recurso).
    - *Subtarea 2.2:* Implementar la lógica en el servicio para recuperar y procesar los datos de reservas desde la base de datos.
    - *Subtarea 2.3:* Optimizar la consulta para garantizar un tiempo de respuesta inferior a 2 segundos.
    - *Subtarea 2.4:* Integrar el endpoint con la base de datos utilizando Prisma y MongoDB.
- **Tarea 3: Desarrollo del Componente de Visualización Interactiva**
    - *Subtarea 3.1:* Seleccionar o desarrollar componentes gráficos (por ejemplo, Chart.js o D3.js) para mostrar los datos.
    - *Subtarea 3.2:* Integrar estos componentes con el endpoint de consulta para actualizar los datos en tiempo real.
    - *Subtarea 3.3:* Implementar opciones de filtrado y ordenamiento en la interfaz.
- **Tarea 4: Registro de Auditoría y Manejo de Excepciones**
    - *Subtarea 4.1:* Configurar logging para registrar cada generación y actualización del reporte.
    - *Subtarea 4.2:* Implementar manejo de excepciones para capturar errores en el flujo de consulta.
- **Tarea 5: Pruebas y Documentación**
    - *Subtarea 5.1:* Desarrollar pruebas unitarias e integración con Jasmine utilizando escenarios BDD (Given-When-Then).
    - *Subtarea 5.2:* Documentar la funcionalidad en la guía del usuario y en la documentación técnica.
    - *Subtarea 5.3:* Integrar la funcionalidad en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

SubHU-27.2: Exportación del Reporte a CSV

Historia de Usuario

Como **Administrador**, quiero **exportar el reporte interactivo de reservas a un archivo CSV** para poder compartir y analizar los datos en herramientas externas y facilitar la toma de decisiones.

Criterios de Aceptación

- La interfaz debe incluir un botón “Exportar a CSV” en la vista del reporte interactivo.
- El archivo CSV debe incluir todos los datos mostrados en el reporte, respetando el formato adecuado.
- La exportación debe completarse en menos de 2 segundos bajo condiciones normales.

- La acción de exportación debe quedar registrada en el historial de auditoría.
- El archivo CSV debe poder descargarse sin errores.

Tareas y Subtareas

- **Tarea 1: Desarrollo del Endpoint de Exportación a CSV**
  - *Subtarea 1.1:* Definir el DTO para la exportación, que incluya los mismos filtros aplicados en la visualización del reporte.
  - *Subtarea 1.2:* Implementar la lógica en el servicio para generar el archivo CSV a partir de los datos del reporte.
  - *Subtarea 1.3:* Validar que el CSV contenga todos los campos requeridos y en el formato correcto.
- **Tarea 2: Integración en la Interfaz de Usuario**
  - *Subtarea 2.1:* Añadir el botón “Exportar a CSV” en la pantalla del reporte interactivo.
  - *Subtarea 2.2:* Conectar el botón con el endpoint de exportación para iniciar la descarga.
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
  - *Subtarea 3.1:* Configurar logging para registrar cada acción de exportación.
  - *Subtarea 3.2:* Implementar manejo de excepciones para capturar errores durante la generación o descarga del CSV.
- **Tarea 4: Pruebas y Documentación**
  - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine y escenarios BDD (Given-When-Then) para el flujo de exportación.
  - *Subtarea 4.2:* Documentar el proceso de exportación en la guía del usuario y la documentación técnica.
  - *Subtarea 4.3:* Integrar la funcionalidad en el pipeline de CI/CD.

RF-33: Posibilidad de exportar reportes en formato CSV, indispensable para el análisis externo

HU-28: Exportación de Reportes en Formato CSV

Historia de Usuario

Como **Administrador**, quiero **exportar los reportes generados en formato CSV** para analizar y compartir la información de uso de los recursos en herramientas externas, facilitando la toma de decisiones y el análisis de datos.

Criterios de Aceptación

- El sistema debe ofrecer una opción clara (botón o enlace) en la interfaz de reportes para exportar el reporte actual a CSV.
- El archivo CSV debe incluir todos los campos y datos mostrados en el reporte (por ejemplo, número total de reservas, porcentajes, métricas de utilización, etc.).
- La exportación debe respetar los filtros aplicados (por programa académico, período de tiempo y tipo de recurso).

- El proceso de generación del archivo CSV debe completarse en menos de 2 segundos bajo condiciones normales de carga.
- El usuario debe poder descargar el archivo CSV sin errores.
- La acción de exportación debe registrarse en el historial de auditoría, indicando quién, cuándo y con qué filtros se realizó la exportación.

Para estructurar la solución se desglosa en la siguiente sub-historia:

SubHU-28.1: Desarrollo e Integración del Módulo de Exportación a CSV

Historia de Usuario

Como **Administrador**, quiero **que el sistema genere y me permita descargar el reporte en formato CSV** para disponer de los datos en un formato fácilmente manipulable y compartible.

Criterios de Aceptación

- El módulo debe recibir los filtros aplicados en la vista de reportes y generar un archivo CSV con los datos correspondientes.
- El archivo debe tener el formato correcto, incluyendo encabezados y filas con los datos, y respetar los formatos de fecha y numéricos.
- La generación del CSV se debe realizar automáticamente al hacer clic en “Exportar a CSV”.
- El sistema debe manejar errores en la generación o descarga y notificar al usuario en caso de fallo.
- La acción de exportación se debe registrar en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Exportación**
  - *Subtarea 1.1:* Crear wireframes y mockups que muestren la opción “Exportar a CSV” en la vista de reportes.
  - *Subtarea 1.2:* Validar el diseño con administradores y ajustar según retroalimentación.
- **Tarea 2: Desarrollo del Endpoint de Exportación a CSV**
  - *Subtarea 2.1:* Definir el DTO que incluya los filtros actuales (programa, período, tipo de recurso).
  - *Subtarea 2.2:* Implementar la lógica en el servicio para extraer los datos filtrados desde la base de datos (usando Prisma y MongoDB).
  - *Subtarea 2.3:* Utilizar una librería (por ejemplo, `json2csv` o similar) para transformar los datos en un archivo CSV.
  - *Subtarea 2.4:* Implementar validaciones para asegurar la integridad de los datos exportados.
- **Tarea 3: Integración en el Frontend**
  - *Subtarea 3.1:* Añadir un botón “Exportar a CSV” en la interfaz de reportes.
  - *Subtarea 3.2:* Conectar el botón con el endpoint de exportación para iniciar la descarga.

    - *Subtarea 3.3:* Mostrar mensajes de éxito o error al usuario según corresponda.
- **Tarea 4: Registro de Auditoría y Manejo de Excepciones**
    - *Subtarea 4.1:* Configurar logging (por ejemplo, con Winston) para registrar cada exportación, incluyendo filtros aplicados y usuario.
    - *Subtarea 4.2:* Implementar manejo de excepciones que capture y notifique errores durante la generación o descarga del CSV.
- **Tarea 5: Pruebas y Documentación**
    - *Subtarea 5.1:* Desarrollar pruebas unitarias e integración en Jasmine utilizando escenarios BDD (Given-When-Then) para el flujo de exportación.
    - *Subtarea 5.2:* Documentar el proceso de exportación, incluyendo ejemplos de uso y formatos esperados, en la guía del usuario.
    - *Subtarea 5.3:* Incluir esta funcionalidad en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

Should - Módulo de Reportes y Análisis

RF-34: Registro de feedback por parte de los usuarios, lo que permitirá mejorar la experiencia a lo largo del tiempo

HU-29: Registro de Feedback de Usuarios sobre la Calidad del Servicio

Historia de Usuario

Como **Usuario**, quiero **enviar feedback sobre la calidad del servicio** para que la administración pueda identificar áreas de mejora y optimizar la experiencia de uso de la plataforma.

Criterios de Aceptación

- El sistema debe ofrecer un formulario accesible para que el usuario ingrese su feedback, que incluya:
    - Una calificación numérica (por ejemplo, de 1 a 5).
    - Un campo de comentarios (opcional o obligatorio según configuración).
    - La fecha se registrará automáticamente.
- Se debe validar que la calificación esté dentro del rango permitido (por ejemplo, 1 a 5).
- El feedback enviado se debe almacenar en la base de datos con identificación del usuario, calificación, comentarios y fecha de envío.
- El sistema debe enviar una confirmación visual al usuario de que su feedback fue registrado exitosamente.
- Toda acción de envío de feedback debe quedar registrada en el historial de auditoría.
- El proceso completo (envío, validación y almacenamiento) debe completarse en menos de 2 segundos bajo condiciones normales de carga.

Para cubrir la funcionalidad de registro de feedback, se desglosa en dos sub-historias:

SubHU-29.1: Envío de Feedback por Parte del Usuario

Historia de Usuario

Como **Usuario**, quiero **enviar feedback sobre mi experiencia de servicio** para contribuir a la mejora continua de la plataforma.

Criterios de Aceptación

- El formulario de feedback debe permitir ingresar una calificación y comentarios.
- Se debe validar que la calificación esté dentro del rango establecido.
- Al enviar el feedback, el sistema debe mostrar un mensaje de confirmación.
- El feedback se debe almacenar de forma segura y asociarse al usuario.
- La acción de envío debe registrarse en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Envío de Feedback**
    - *Subtarea 1.1:* Crear wireframes y mockups del formulario de feedback (calificación, comentarios).
    - *Subtarea 1.2:* Validar el diseño con usuarios y stakeholders.
- **Tarea 2: Desarrollo del Endpoint de Envío de Feedback**
    - *Subtarea 2.1:* Definir el DTO para el feedback que incluya la calificación y los comentarios.
    - *Subtarea 2.2:* Implementar la lógica en el servicio para recibir y validar el feedback.
    - *Subtarea 2.3:* Integrar el endpoint con la base de datos usando Prisma y MongoDB.
    - *Subtarea 2.4:* Incluir validaciones para que la calificación esté en el rango permitido (1-5).
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
    - *Subtarea 3.1:* Configurar logging (por ejemplo, con Winston) para registrar cada envío de feedback.
    - *Subtarea 3.2:* Implementar manejo de excepciones para notificar al usuario y administrador en caso de error.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración con Jasmine utilizando escenarios BDD (Given-When-Then) para el flujo de envío.
    - *Subtarea 4.2:* Documentar la funcionalidad en la guía del usuario y en la documentación técnica.
    - *Subtarea 4.3:* Incluir el módulo en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

SubHU-29.2: Consulta y Análisis del Feedback de Usuarios

Historia de Usuario

Como **Administrador**, quiero **consultar y analizar el feedback enviado por los usuarios** para evaluar la calidad del servicio y detectar oportunidades de mejora en la plataforma.

Criterios de Aceptación

- La interfaz de consulta debe mostrar una lista detallada del feedback, incluyendo usuario, calificación, comentarios y fecha.
- Se deben incluir filtros por fecha, usuario y rango de calificación.
- La consulta debe responder en menos de 2 segundos en condiciones normales.
- Se debe ofrecer la opción de exportar los resultados a formato CSV.
- El acceso a la consulta del feedback debe estar restringido a usuarios autorizados.
- Todas las consultas y exportaciones deben quedar registradas en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Consulta del Feedback**

    - *Subtarea 1.1:* Crear wireframes y mockups de la pantalla de consulta y análisis de feedback, con filtros y opción de exportación.
    - *Subtarea 1.2:* Validar el diseño con administradores y equipo de análisis.
- **Tarea 2: Desarrollo del Endpoint de Consulta de Feedback**
    - *Subtarea 2.1:* Definir el DTO para la consulta que incluya los filtros (fecha, usuario, rango de calificación).
    - *Subtarea 2.2:* Implementar la lógica en el servicio para recuperar y filtrar los registros de feedback.
    - *Subtarea 2.3:* Optimizar la consulta para asegurar un tiempo de respuesta inferior a 2 segundos.
    - *Subtarea 2.4:* Integrar el endpoint con la base de datos mediante Prisma y MongoDB.
- **Tarea 3: Desarrollo del Módulo de Exportación a CSV**
    - *Subtarea 3.1:* Implementar la lógica para generar un archivo CSV a partir de los datos filtrados.
    - *Subtarea 3.2:* Integrar la opción de exportación en la interfaz de consulta.
- **Tarea 4: Registro de Auditoría y Manejo de Excepciones**
    - *Subtarea 4.1:* Configurar logging para registrar cada acción de consulta y exportación.
    - *Subtarea 4.2:* Implementar manejo de errores para capturar y notificar fallos en la consulta o exportación.
- **Tarea 5: Pruebas y Documentación**
    - *Subtarea 5.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine con escenarios BDD (Given-When-Then).
    - *Subtarea 5.2:* Documentar la funcionalidad de consulta y análisis del feedback en la guía del usuario.
    - *Subtarea 5.3:* Integrar el módulo en el pipeline de CI/CD.

RF-35: Evaluación de usuarios por parte del personal administrativo, para incentivar el uso responsable

HU-30: Evaluación de Usuarios por el Personal Administrativo

Historia de Usuario

Como **Personal Administrativo**, quiero **evaluar el desempeño y comportamiento de los usuarios en el uso de los recursos** para identificar oportunidades de mejora, fomentar un uso responsable y optimizar la asignación de los recursos institucionales.

Criterios de Aceptación

- Se debe disponer de un formulario de evaluación accesible para el personal administrativo.
- El formulario incluirá campos para puntuar aspectos clave (por ejemplo, puntualidad, cumplimiento de reservas, incidencias reportadas, etc.) con un rango numérico definido (por ejemplo, 1 a 5).
- Se debe permitir agregar comentarios adicionales para justificar o complementar la evaluación.

- Al enviar la evaluación, el sistema debe asociar la evaluación al usuario evaluado y registrar la fecha de la evaluación.
- Los resultados de la evaluación (puntaje y comentarios) deben almacenarse de forma inalterable en la base de datos.
- La evaluación registrada debe actualizar un indicador o score que permita comparar el uso responsable entre usuarios.
- Se debe notificar al evaluador la confirmación del registro y registrar la acción en el historial de auditoría.
- La funcionalidad debe integrarse con el módulo de reservas para vincular evaluaciones con incidencias o patrones de uso.

Para cubrir integralmente la funcionalidad, se desglosa en dos sub-historias:

SubHU-30.1: Registro de Evaluación de Usuarios

Historia de Usuario

Como **Personal Administrativo**, quiero **registrar evaluaciones de los usuarios mediante un formulario estructurado** para documentar su comportamiento en el uso de los recursos y disponer de información para tomar decisiones correctivas.

Criterios de Aceptación

- El formulario debe permitir ingresar:
    - Un puntaje numérico (rango 1-5) para cada criterio evaluado.
    - Comentarios adicionales (opcional o obligatorio, según configuración).
    - La fecha de evaluación se registrará automáticamente.
- El sistema validará que los puntajes ingresados estén dentro del rango permitido.
- La evaluación se asociará al perfil del usuario evaluado.
- Al enviar el formulario, se mostrará un mensaje de confirmación de registro.
- La acción de registro se documentará en el historial de auditoría con la identificación del evaluador, fecha y detalles de la evaluación.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Evaluación**
    - *Subtarea 1.1:* Crear wireframes y mockups del formulario de evaluación que incluya campos para puntaje y comentarios.
    - *Subtarea 1.2:* Validar el diseño con el personal administrativo y ajustar según feedback.
- **Tarea 2: Desarrollo del Endpoint de Registro de Evaluaciones en NestJS**
    - *Subtarea 2.1:* Definir el DTO para la evaluación (incluyendo puntajes, comentarios y datos del usuario).
    - *Subtarea 2.2:* Implementar la lógica en el servicio para almacenar la evaluación en la base de datos (MongoDB mediante Prisma).
    - *Subtarea 2.3:* Incluir validaciones para asegurar que los puntajes se encuentren dentro del rango 1-5.
- **Tarea 3: Registro de Auditoría y Manejo de Errores**

    - *Subtarea 3.1:* Configurar logging (por ejemplo, utilizando Winston) para registrar la acción de evaluación.
    - *Subtarea 3.2:* Implementar manejo de excepciones para notificar errores en el proceso de registro.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración con Jasmine, utilizando escenarios BDD (Given-When-Then).
    - *Subtarea 4.2:* Documentar la funcionalidad en la guía del usuario y la documentación técnica.
    - *Subtarea 4.3:* Integrar el módulo en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

SubHU-30.2: Consulta y Análisis de Evaluaciones

Historia de Usuario

Como **Personal Administrativo**, quiero **consultar y analizar las evaluaciones de los usuarios** para identificar patrones de uso responsable o inadecuado y tomar acciones correctivas que optimicen la asignación de los recursos.

Criterios de Aceptación

- La interfaz debe permitir visualizar un historial de evaluaciones que incluya:
    - Datos del usuario evaluado.
    - Puntajes y comentarios de cada evaluación.
    - Fecha de cada evaluación.
- Se deben ofrecer filtros por usuario, fecha y rango de puntajes.
- La consulta debe responder en menos de 2 segundos en condiciones normales.
- Se debe permitir exportar el historial de evaluaciones en formato CSV.
- Toda acción de consulta y exportación debe quedar registrada en el historial de auditoría.
- El acceso a esta información debe estar restringido a personal autorizado.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Consulta de Evaluaciones**
    - *Subtarea 1.1:* Crear wireframes y mockups para la pantalla de consulta, incluyendo filtros y opción de exportación.
    - *Subtarea 1.2:* Validar el diseño con administradores y el equipo de análisis.
- **Tarea 2: Desarrollo del Endpoint de Consulta de Evaluaciones en NestJS**
    - *Subtarea 2.1:* Definir el DTO para la consulta (incluyendo filtros por usuario, fecha y puntaje).
    - *Subtarea 2.2:* Implementar la lógica en el servicio para recuperar y formatear las evaluaciones.
    - *Subtarea 2.3:* Optimizar la consulta para asegurar un tiempo de respuesta inferior a 2 segundos.
    - *Subtarea 2.4:* Integrar el endpoint con la base de datos utilizando Prisma y MongoDB.
- **Tarea 3: Desarrollo del Módulo de Exportación a CSV para Evaluaciones**

    - *Subtarea 3.1:* Implementar la lógica para generar un archivo CSV con los datos filtrados.
    - *Subtarea 3.2:* Integrar la opción de exportación en la interfaz.
- **Tarea 4: Registro de Auditoría y Manejo de Excepciones**
    - *Subtarea 4.1:* Configurar logging para registrar cada acción de consulta y exportación.
    - *Subtarea 4.2:* Implementar manejo de errores para capturar y notificar incidencias en la consulta.
- **Tarea 5: Pruebas y Documentación**
    - *Subtarea 5.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine con escenarios BDD (Given-When-Then).
    - *Subtarea 5.2:* Documentar el proceso de consulta y exportación en la guía del usuario.
    - *Subtarea 5.3:* Integrar la funcionalidad en el pipeline de CI/CD.

Could - Módulo de Reportes y Análisis

RF-36: Dashboards interactivos con estadísticas en tiempo real, que pueden implementarse en fases complementarias

HU-31: Visualización de Dashboards Interactivos en Tiempo Real

Historia de Usuario

Como **Administrador**, quiero **visualizar dashboards interactivos con estadísticas en tiempo real** para monitorear el uso de recursos y tomar decisiones basadas en datos actualizados, mejorando la eficiencia en la asignación y optimización de recursos.

Criterios de Aceptación

- La pantalla de dashboards debe mostrar indicadores clave como número total de reservas, tasa de cancelación, utilización por recurso, distribución por programa académico y otros KPIs relevantes.
- Los datos deben actualizarse en tiempo real o con un intervalo configurado (por ejemplo, cada 30 segundos), con un tiempo de respuesta inferior a 2 segundos en condiciones normales.
- Se debe permitir aplicar filtros por fecha, tipo de recurso, programa académico y ubicación.
- La visualización debe incluir gráficos interactivos (barras, líneas, pastel) y tablas comparativas.
- La interfaz debe ser responsiva y funcionar en dispositivos móviles y de escritorio.
- Cada acción (aplicación de filtros, actualización de datos) debe quedar registrada en el historial de auditoría.

Para cubrir de forma integral este requerimiento se desglosa en dos sub-historias:

SubHU-31.1: Visualización del Dashboard Interactivo

Historia de Usuario

Como **Administrador**, quiero **ver un dashboard interactivo que muestre estadísticas actualizadas en tiempo real** para obtener una visión clara y dinámica del uso de recursos y detectar rápidamente áreas de mejora.

Criterios de Aceptación

- La pantalla debe mostrar gráficos y tablas con indicadores clave (número total de reservas, uso por tipo de recurso, etc.).
- Se deben aplicar filtros dinámicos (por fecha, programa, recurso, ubicación) que actualicen la visualización de forma instantánea.
- La actualización de los datos debe ocurrir en tiempo real o con un intervalo configurado (ej. cada 30 segundos).

- La interfaz debe ser intuitiva, permitiendo al usuario interactuar con los gráficos (por ejemplo, haciendo clic para ver detalles).
- La generación de la visualización debe quedar registrada en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz del Dashboard**
  - *Subtarea 1.1:* Crear wireframes y mockups que muestren la disposición de gráficos, tablas e indicadores.
  - *Subtarea 1.2:* Validar el diseño con administradores y usuarios clave.
  - *Subtarea 1.3:* Definir la simbología visual (colores, íconos) para cada indicador.
- **Tarea 2: Desarrollo del Componente de Visualización (Frontend)**
  - *Subtarea 2.1:* Seleccionar una librería de gráficos (por ejemplo, Chart.js, D3.js) adecuada para la representación interactiva.
  - *Subtarea 2.2:* Implementar el componente del dashboard integrando gráficos y tablas.
  - *Subtarea 2.3:* Programar la actualización dinámica de datos utilizando WebSockets o técnicas de polling.
- **Tarea 3: Desarrollo del Endpoint de Consulta de Datos (Backend)**
  - *Subtarea 3.1:* Definir el DTO que reciba filtros (fecha, recurso, programa, ubicación).
  - *Subtarea 3.2:* Implementar la lógica en el servicio para recuperar estadísticas y datos relevantes de la base de datos (MongoDB con Prisma).
  - *Subtarea 3.3:* Optimizar la consulta para garantizar una respuesta en menos de 2 segundos.
- **Tarea 4: Registro de Auditoría y Manejo de Excepciones**
  - *Subtarea 4.1:* Configurar logging para registrar cada consulta y actualización en el dashboard.
  - *Subtarea 4.2:* Implementar manejo de excepciones para notificar y registrar errores en la actualización de datos.
- **Tarea 5: Pruebas y Documentación**
  - *Subtarea 5.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine y escenarios BDD (Given-When-Then).
  - *Subtarea 5.2:* Documentar la funcionalidad del dashboard en la guía del usuario y en la documentación técnica.
  - *Subtarea 5.3:* Integrar la funcionalidad en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

SubHU-31.2: Configuración de Filtros y Exportación de Datos del Dashboard

Historia de Usuario

Como **Administrador**, quiero **configurar filtros avanzados y exportar los datos mostrados en el dashboard a formato CSV** para realizar análisis externos y compartir información de forma sencilla.

Criterios de Aceptación

- La interfaz debe permitir aplicar filtros (por fecha, programa académico, tipo de recurso, ubicación) que se reflejen en el dashboard.
- Se debe incluir un botón “Exportar a CSV” que genere un archivo con los datos filtrados del dashboard.
- El archivo CSV debe incluir encabezados y todos los datos relevantes (indicadores, resultados, etc.) en el formato correcto.
- La exportación debe completarse en menos de 2 segundos y permitir la descarga sin errores.
- La acción de exportación y cualquier cambio en los filtros debe quedar registrada en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Funcionalidad de Filtros y Exportación**
    - *Subtarea 1.1:* Actualizar los wireframes del dashboard para incluir opciones de filtrado avanzado.
    - *Subtarea 1.2:* Diseñar la interfaz del botón “Exportar a CSV” y la presentación del archivo generado.
    - *Subtarea 1.3:* Validar el diseño con administradores y usuarios clave.
- **Tarea 2: Desarrollo del Módulo de Filtrado (Backend y Frontend)**
    - *Subtarea 2.1:* Ampliar el DTO y la lógica del endpoint de consulta para aceptar filtros adicionales.
    - *Subtarea 2.2:* Implementar la integración de los filtros en el componente de visualización.
    - *Subtarea 2.3:* Asegurar que la actualización del dashboard responda a los cambios en los filtros.
- **Tarea 3: Desarrollo del Endpoint de Exportación a CSV**
    - *Subtarea 3.1:* Definir el DTO para la exportación que incluya los filtros aplicados.
    - *Subtarea 3.2:* Implementar la lógica en el servicio para generar el archivo CSV a partir de los datos filtrados.
    - *Subtarea 3.3:* Validar el formato del CSV (encabezados, datos correctos).
- **Tarea 4: Registro de Auditoría y Manejo de Excepciones**
    - *Subtarea 4.1:* Configurar logging para registrar cada acción de filtrado y exportación.
    - *Subtarea 4.2:* Implementar manejo de errores para notificar fallos en la exportación y registrarlos.
- **Tarea 5: Pruebas y Documentación**
    - *Subtarea 5.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine y escenarios BDD (Given-When-Then) para el flujo de filtrado y exportación.
    - *Subtarea 5.2:* Documentar la funcionalidad de filtros y exportación en la guía del usuario y la documentación técnica.
    - *Subtarea 5.3:* Integrar la funcionalidad en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

RF-37: Reporte de demanda insatisfecha, útil para identificar áreas de mejora, pero no crítico en el lanzamiento inicial

HU-32: Generación de Reporte de Demanda Insatisfecha

Historia de Usuario

Como **Administrador**, quiero **generar un reporte de demanda insatisfecha** para identificar períodos y recursos donde la demanda de reservas no se ha satisfecho, permitiéndome tomar acciones correctivas para optimizar la asignación y disponibilidad de recursos.

Criterios de Aceptación

- El reporte debe mostrar, por cada recurso, los períodos con alta demanda en los que no se concretaron reservas.
- Debe incluir indicadores como:
    - Número de solicitudes de reserva rechazadas o no completadas.
    - Tasa de demanda insatisfecha (porcentaje de solicitudes no atendidas frente a la demanda total).
    - Datos específicos del recurso (nombre, tipo, ubicación) y del período (fecha y hora).
- La generación del reporte debe completarse en menos de 2 segundos bajo condiciones normales.
- La interfaz debe permitir aplicar filtros (por recurso, fecha, tipo de usuario) para refinar la información.
- Se debe permitir exportar el reporte en formato CSV.
- Todas las acciones (generación, consulta y exportación del reporte) deben quedar registradas en el historial de auditoría.

Para abordar esta funcionalidad se desglosa en dos sub-historias:

SubHU-32.1: Visualización Interactiva del Reporte de Demanda Insatisfecha

Historia de Usuario

Como **Administrador**, quiero **visualizar un dashboard interactivo que muestre la demanda insatisfecha de reservas** para identificar rápidamente los recursos y períodos críticos y facilitar la toma de decisiones.

Criterios de Aceptación

- La interfaz debe mostrar gráficos y tablas que ilustren:
    - El número total de solicitudes insatisfechas por recurso.
    - La tasa de demanda insatisfecha en un rango de fechas.
- Se deben poder aplicar filtros por recurso, período de tiempo y tipo de usuario.
- La información visualizada debe actualizarse en tiempo real o mediante acción de “refrescar” en menos de 2 segundos.
- La visualización debe ser responsiva y fácil de interpretar.

- La acción de generación y actualización del reporte debe quedar registrada en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz del Dashboard de Demanda Insatisfecha**
  - *Subtarea 1.1:* Crear wireframes y mockups que muestren gráficos (barras, líneas o pastel) y tablas con los indicadores de demanda insatisfecha.
  - *Subtarea 1.2:* Validar el diseño con administradores y ajustarlo según retroalimentación.
- **Tarea 2: Desarrollo del Endpoint de Consulta de Demanda Insatisfecha (Backend)**
  - *Subtarea 2.1:* Definir el DTO que reciba filtros (recurso, fecha, tipo de usuario).
  - *Subtarea 2.2:* Implementar la lógica en el servicio para extraer de la base de datos (MongoDB con Prisma) los datos de reservas y solicitudes no atendidas.
  - *Subtarea 2.3:* Optimizar la consulta para asegurar una respuesta en menos de 2 segundos.
- **Tarea 3: Desarrollo del Componente de Visualización Interactiva (Frontend)**
  - *Subtarea 3.1:* Seleccionar una librería de gráficos interactivos (por ejemplo, Chart.js o D3.js).
  - *Subtarea 3.2:* Integrar el componente con el endpoint de consulta para actualizar los datos en tiempo real o mediante “refresco”.
  - *Subtarea 3.3:* Implementar opciones de filtrado y ordenamiento en la interfaz.
- **Tarea 4: Registro de Auditoría y Manejo de Excepciones**
  - *Subtarea 4.1:* Configurar logging para registrar cada generación y actualización del dashboard.
  - *Subtarea 4.2:* Implementar manejo de errores y excepciones que notifiquen al administrador en caso de fallos.
- **Tarea 5: Pruebas y Documentación**
  - *Subtarea 5.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine con escenarios BDD (Given-When-Then).
  - *Subtarea 5.2:* Documentar la funcionalidad en la guía del usuario y la documentación técnica.
  - *Subtarea 5.3:* Integrar el módulo en el pipeline de CI/CD.

SubHU-32.2: Exportación del Reporte de Demanda Insatisfecha a CSV

Historia de Usuario

Como **Administrador**, quiero **exportar el reporte de demanda insatisfecha en formato CSV** para poder analizar y compartir los datos con otros departamentos y realizar análisis externos con herramientas especializadas.

Criterios de Aceptación

- Se debe disponer de un botón “Exportar a CSV” en la interfaz del dashboard.

- El archivo CSV generado debe incluir todos los datos relevantes mostrados en el reporte, con encabezados y en un formato correcto.
- La exportación debe respetar los filtros aplicados en la visualización del reporte.
- El proceso de generación y descarga del CSV debe completarse en menos de 2 segundos.
- La acción de exportación debe quedar registrada en el historial de auditoría.
- El archivo CSV debe ser descargable sin errores.

Tareas y Subtareas

- **Tarea 1: Diseño de la Funcionalidad de Exportación**
    - *Subtarea 1.1:* Crear mockups de la opción “Exportar a CSV” en la pantalla del dashboard.
    - *Subtarea 1.2:* Validar el diseño con administradores y ajustar según retroalimentación.
- **Tarea 2: Desarrollo del Endpoint de Exportación a CSV (Backend)**
    - *Subtarea 2.1:* Definir el DTO para la exportación que incluya los filtros aplicados.
    - *Subtarea 2.2:* Implementar la lógica en el servicio para extraer los datos filtrados y convertirlos a CSV utilizando una librería (por ejemplo, json2csv).
    - *Subtarea 2.3:* Validar el formato del archivo CSV y asegurarse de que contenga todos los campos requeridos.
- **Tarea 3: Integración en el Frontend**
    - *Subtarea 3.1:* Añadir el botón “Exportar a CSV” en la interfaz del dashboard.
    - *Subtarea 3.2:* Conectar el botón con el endpoint de exportación para iniciar la descarga.
    - *Subtarea 3.3:* Mostrar mensajes de éxito o error al usuario según corresponda.
- **Tarea 4: Registro de Auditoría y Manejo de Excepciones**
    - *Subtarea 4.1:* Configurar logging para registrar cada exportación realizada, incluyendo los filtros utilizados y el usuario que la solicitó.
    - *Subtarea 4.2:* Implementar manejo de excepciones para capturar y notificar errores en el proceso de exportación.
- **Tarea 5: Pruebas y Documentación**
    - *Subtarea 5.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine y escenarios BDD (Given-When-Then) para el flujo de exportación.
    - *Subtarea 5.2:* Documentar la funcionalidad en la guía del usuario y en la documentación técnica.
    - *Subtarea 5.3:* Integrar el módulo de exportación en el pipeline de CI/CD.

Must - Módulo de Auth (Seguridad y Control de Accesos)

RF-41: Gestión de roles y permisos según el perfil del usuario, vital para la seguridad

HU-33: Gestión de Roles y Permisos según el Perfil del Usuario

Historia de Usuario

Como **Administrador**, quiero **gestionar roles y permisos según el perfil del usuario** para asegurar que solo los usuarios autorizados puedan acceder y modificar información sensible, garantizando la seguridad y el cumplimiento de las políticas institucionales.

Criterios de Aceptación

- La interfaz debe permitir crear, editar y eliminar roles.
- Se debe poder asignar un conjunto de permisos (por ejemplo, lectura, escritura, modificación, eliminación) a cada rol.
- La funcionalidad debe permitir ver un listado de roles existentes, con sus permisos asignados, de forma clara y ordenada.
- Cualquier acción de creación, modificación o eliminación de roles y permisos debe quedar registrada en el historial de auditoría (incluyendo el usuario que realizó la acción, fecha, hora y cambios efectuados).
- Los cambios en roles y permisos deben aplicarse de forma inmediata y reflejarse en el control de acceso del sistema.
- Se deben manejar adecuadamente los errores (por ejemplo, no permitir duplicidad de roles o asignar permisos inválidos) y notificar al usuario en caso de fallo.

Para estructurar la solución, se desglosa en dos sub-historias:

SubHU-33.1: Gestión de Roles (Creación, Edición y Eliminación)

Historia de Usuario

Como **Administrador**, quiero **crear, editar y eliminar roles** para definir y actualizar las categorías de usuarios que tendrán distintos niveles de acceso en el sistema.

Criterios de Aceptación

- La interfaz debe permitir ingresar un nombre único para cada rol.
- Se debe validar que no existan roles duplicados.
- El sistema debe permitir editar el nombre y los detalles de un rol existente.
- Se debe permitir la eliminación de roles, salvo aquellos que estén en uso activo (mostrando un mensaje de error o solicitando confirmación adicional).
- Cada acción (creación, edición, eliminación) debe quedar registrada en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Gestión de Roles**
  - *Subtarea 1.1:* Crear wireframes y mockups para la pantalla de administración de roles, incluyendo formularios para crear y editar.
  - *Subtarea 1.2:* Validar el diseño con usuarios clave y ajustar según retroalimentación.
- **Tarea 2: Desarrollo del Endpoint para Gestión de Roles en NestJS**
  - *Subtarea 2.1:* Definir el DTO para la creación y edición de roles.
  - *Subtarea 2.2:* Implementar la lógica en el servicio para crear, editar y eliminar roles, incluyendo validación de duplicidad.
  - *Subtarea 2.3:* Integrar el endpoint con la base de datos utilizando Prisma y MongoDB.
  - *Subtarea 2.4:* Desarrollar pruebas unitarias para validar la funcionalidad.
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
  - *Subtarea 3.1:* Configurar logging (por ejemplo, usando Winston) para registrar cada acción sobre roles.
  - *Subtarea 3.2:* Implementar manejo de errores y notificaciones en caso de fallos (por ejemplo, intento de eliminar un rol en uso).
- **Tarea 4: Pruebas de Integración y Documentación**
  - *Subtarea 4.1:* Desarrollar pruebas de integración utilizando Jasmine (escenarios Given-When-Then).
  - *Subtarea 4.2:* Documentar la funcionalidad en la guía del usuario y la documentación técnica.
  - *Subtarea 4.3:* Integrar el módulo de roles en el pipeline de CI/CD.

SubHU-33.2: Asignación y Gestión de Permisos por Rol

Historia de Usuario

Como **Administrador**, quiero **asignar y gestionar permisos específicos para cada rol** para asegurar que cada categoría de usuario tenga los niveles de acceso adecuados y restringir la modificación de información sensible.

Criterios de Aceptación

- La interfaz debe permitir asignar permisos (por ejemplo, lectura, escritura, edición, eliminación) a cada rol.
- Se debe poder modificar los permisos asignados a un rol existente.
- La asignación de permisos debe validarse para asegurar que solo se puedan asignar permisos predefinidos y válidos.
- Los cambios en la asignación de permisos deben reflejarse inmediatamente en el sistema de control de acceso.
- Toda acción (asignación, modificación o eliminación de permisos) debe quedar registrada en el historial de auditoría.
- Se deben manejar errores (por ejemplo, asignar permisos inexistentes) y notificar al usuario en caso de fallo.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz para Gestión de Permisos**
  - *Subtarea 1.1:* Crear wireframes y mockups que muestren cómo se asignarán y visualizarán los permisos para cada rol.
  - *Subtarea 1.2:* Validar el diseño con administradores y usuarios clave.
- **Tarea 2: Desarrollo del Endpoint para Asignación de Permisos en NestJS**
  - *Subtarea 2.1:* Definir el DTO para la asignación y modificación de permisos.
  - *Subtarea 2.2:* Implementar la lógica en el servicio para asignar, editar y eliminar permisos para un rol.
  - *Subtarea 2.3:* Integrar validaciones para asegurar que solo se asignen permisos válidos y predefinidos.
  - *Subtarea 2.4:* Integrar el endpoint con la base de datos utilizando Prisma y MongoDB.
  - *Subtarea 2.5:* Desarrollar pruebas unitarias para esta funcionalidad.
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
  - *Subtarea 3.1:* Configurar logging para registrar todas las acciones de asignación y modificación de permisos.
  - *Subtarea 3.2:* Implementar manejo de excepciones para notificar y registrar cualquier error en el proceso.
- **Tarea 4: Pruebas de Integración y Documentación**
  - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine y escenarios BDD (Given-When-Then).
  - *Subtarea 4.2:* Documentar el proceso de gestión de permisos en la guía del usuario y la documentación técnica.
  - *Subtarea 4.3:* Integrar la funcionalidad en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

RF-42: Restricción de modificaciones a los recursos únicamente a administradores, para proteger la integridad de la información

HU-34: Restricción de Modificaciones a los Recursos Solo para Administradores

Historia de Usuario

Como **Administrador**, quiero que **solo los usuarios con rol de administrador puedan modificar los recursos** para garantizar la integridad de la información y evitar cambios no autorizados.

Criterios de Aceptación

- Al intentar modificar un recurso, el sistema debe verificar el rol del usuario autenticado.
- Si el usuario es administrador, se permitirá la modificación y se actualizará el recurso.

- Si el usuario no es administrador, el sistema deberá rechazar la operación y devolver un mensaje claro (por ejemplo, "No tiene permisos para modificar este recurso") con un código de error 403 (Forbidden).
- Todas las acciones (modificaciones aprobadas o intentos rechazados) deben quedar registradas en el historial de auditoría, incluyendo quién realizó la acción, la fecha y el resultado.
- La verificación del rol y la restricción deben aplicarse tanto en el backend (endpoint) como reflejarse en la interfaz de usuario (por ejemplo, ocultando o deshabilitando botones de edición para usuarios no autorizados).

Para cubrir la funcionalidad integralmente se desglosa en dos sub-historias:

SubHU-34.1: Permitir Modificaciones Sólo a Administradores

Historia de Usuario

Como **Administrador**, quiero que **el sistema permita la modificación de recursos únicamente a usuarios con rol de administrador** para asegurar que solo personal autorizado realice cambios en la información.

Criterios de Aceptación

- El endpoint de modificación de recursos debe validar que el usuario tenga rol de administrador.
- Si la validación es exitosa, se debe proceder con la modificación y actualizar la información en la base de datos.
- La respuesta exitosa debe incluir un mensaje de confirmación y los datos actualizados del recurso.
- La acción de modificación aprobada debe registrarse en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño y Especificación del Requisito**
  - *Subtarea 1.1:* Revisar y documentar el flujo de modificación de recursos en el sistema.
  - *Subtarea 1.2:* Definir claramente las reglas de negocio que permiten la modificación solo a administradores.
- **Tarea 2: Desarrollo del Endpoint de Modificación con Validación de Rol**
  - *Subtarea 2.1:* Definir el DTO para la actualización de recursos.
  - *Subtarea 2.2:* Implementar en el servicio de recursos la lógica de validación que verifique que el usuario autenticado tenga rol de administrador.
  - *Subtarea 2.3:* Integrar la validación en un middleware o guard (por ejemplo, utilizando los Guards de NestJS).
  - *Subtarea 2.4:* Actualizar el recurso en la base de datos si la validación es exitosa, usando Prisma y MongoDB.
- **Tarea 3: Registro de Auditoría**
  - *Subtarea 3.1:* Configurar logging (por ejemplo, con Winston) para registrar cada modificación aprobada, incluyendo el identificador del usuario, fecha y datos modificados.

    - *Subtarea 3.2:* Almacenar la información en el historial de auditoría.
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine (escenarios Given-When-Then) para el flujo de modificación por administradores.
    - *Subtarea 4.2:* Documentar el proceso de modificación y las reglas de validación en la guía del usuario y en la documentación técnica.
    - *Subtarea 4.3:* Integrar la funcionalidad en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

SubHU-34.2: Manejo de Intentos de Modificación por Usuarios No Autorizados

Historia de Usuario

Como **Usuario no Administrador**, quiero que **se me impida modificar los recursos y se me muestre un mensaje de error** para saber que no tengo los permisos necesarios y evitar cambios no autorizados.

Criterios de Aceptación

- Si un usuario no administrador intenta modificar un recurso, el sistema debe retornar un código de error 403 (Forbidden) junto con un mensaje explicativo.
- La interfaz de usuario debe deshabilitar o ocultar las opciones de modificación para usuarios sin permisos.
- El intento de modificación por un usuario no autorizado debe quedar registrado en el historial de auditoría, incluyendo el identificador del usuario y la acción rechazada.
- La validación debe aplicarse de forma consistente tanto en el backend como en el frontend.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz para Usuarios No Administradores**
    - *Subtarea 1.1:* Actualizar la interfaz de recursos para que el botón de "Editar" se oculte o se deshabilite para usuarios sin rol de administrador.
    - *Subtarea 1.2:* Crear mensajes de error claros para mostrar en caso de que un usuario no autorizado intente acceder a la edición mediante URL directa.
- **Tarea 2: Implementación de Validación en el Backend**
    - *Subtarea 2.1:* Configurar un Guard o middleware en el endpoint de modificación que verifique el rol del usuario.
    - *Subtarea 2.2:* Retornar un error 403 con un mensaje adecuado si la validación falla.
    - *Subtarea 2.3:* Desarrollar pruebas unitarias para validar la restricción de acceso en el endpoint.
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
    - *Subtarea 3.1:* Configurar logging para registrar cada intento de modificación fallido por usuarios no autorizados.
    - *Subtarea 3.2:* Implementar manejo de excepciones que asegure que se capture y notifique el intento fallido.
- **Tarea 4: Pruebas y Documentación**

- *Subtarea 4.1:* Desarrollar pruebas de integración con Jasmine utilizando escenarios Given-When-Then para simular intentos no autorizados.
- *Subtarea 4.2:* Documentar la restricción y el comportamiento esperado para usuarios no autorizados en la guía del usuario.
- *Subtarea 4.3:* Integrar la funcionalidad en el pipeline de CI/CD.

RF-43: Implementación de autenticación y autorización mediante credenciales o SSO, asegurando un acceso controlado

HU-35: Implementación de Autenticación y Autorización Segura

Historia de Usuario

Como **Usuario de la plataforma**, quiero **iniciar sesión mediante mis credenciales o a través de un sistema SSO** para acceder de forma segura y centralizada a mis funciones sin tener que gestionar múltiples contraseñas, garantizando la integridad y confidencialidad de mi información.

Criterios de Aceptación

- El sistema debe permitir la autenticación tradicional mediante usuario (correo electrónico o identificador) y contraseña.
- El sistema debe integrarse con un mecanismo de Single Sign-On (SSO) utilizando protocolos estándar (por ejemplo, OAuth2 o SAML) para usuarios universitarios.
- Tras la autenticación, se debe generar un token de acceso seguro (por ejemplo, JWT) con un tiempo de expiración configurable.
- El sistema debe validar y autorizar el acceso basándose en el token emitido, integrándose con el módulo de roles (RF-41).
- Se debe implementar un mecanismo de renovación y revocación de tokens.
- Los accesos, tanto exitosos como fallidos, deben registrarse en el historial de auditoría.
- La interfaz de inicio de sesión y SSO debe ser intuitiva y adaptable a dispositivos móviles y de escritorio.
- La respuesta del sistema (autenticación exitosa o error) debe ocurrir en menos de 2 segundos en condiciones normales de carga.

Para estructurar la solución se desglosa en dos sub-historias:

SubHU-35.1: Autenticación Tradicional mediante Credenciales

Historia de Usuario

Como **Usuario**, quiero **iniciar sesión introduciendo mi usuario y contraseña** para acceder al sistema de manera rápida y segura.

Criterios de Aceptación

- La interfaz debe mostrar un formulario de login con campos para usuario (correo electrónico o ID) y contraseña.

- El sistema validará las credenciales contra la base de datos de usuarios.
- En caso de autenticación exitosa, se generará un token JWT y se redirigirá al usuario a la página principal.
- Si las credenciales son inválidas, se mostrará un mensaje de error claro y se registrará el intento fallido.
- Todas las acciones de login (éxito o error) se deben registrar en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Login**
  - *Subtarea 1.1:* Crear wireframes y mockups del formulario de inicio de sesión.
  - *Subtarea 1.2:* Validar el diseño con usuarios y stakeholders.
- **Tarea 2: Desarrollo del Endpoint de Autenticación**
  - *Subtarea 2.1:* Definir el DTO para el login (usuario y contraseña).
  - *Subtarea 2.2:* Implementar la lógica en el servicio de autenticación en NestJS, utilizando Passport.js (o similar) para validar credenciales.
  - *Subtarea 2.3:* Integrar con la base de datos de usuarios usando Prisma y MongoDB.
  - *Subtarea 2.4:* Generar y emitir un token JWT seguro tras la validación.
- **Tarea 3: Registro de Auditoría y Manejo de Errores**
  - *Subtarea 3.1:* Configurar logging (ej., Winston) para registrar cada intento de login.
  - *Subtarea 3.2:* Implementar manejo de excepciones para capturar y reportar errores de autenticación.
- **Tarea 4: Pruebas y Documentación**
  - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración con Jasmine (escenarios Given-When-Then) para el flujo de login.
  - *Subtarea 4.2:* Documentar la funcionalidad en la guía del usuario y en la documentación técnica.
  - *Subtarea 4.3:* Incluir el módulo de autenticación en el pipeline de CI/CD.

SubHU-35.2: Integración con SSO (Single Sign-On)

Historia de Usuario

Como **Usuario universitario**, quiero **iniciar sesión utilizando un sistema SSO** para acceder a la plataforma sin tener que recordar credenciales adicionales, aprovechando la infraestructura de autenticación centralizada de la universidad.

Criterios de Aceptación

- El sistema debe ofrecer una opción para iniciar sesión mediante SSO.
- La integración SSO debe utilizar protocolos estándar (por ejemplo, OAuth2 o SAML) y validar la identidad del usuario contra el proveedor institucional.
- Tras la autenticación SSO, el sistema debe mapear los datos del usuario al modelo interno y generar un token JWT.
- El flujo SSO debe ser transparente para el usuario y completar la autenticación en menos de 2 segundos.

- Los accesos mediante SSO deben quedar registrados en el historial de auditoría, incluyendo detalles del proveedor SSO.
- Se debe manejar correctamente cualquier error en la autenticación SSO, notificando al usuario y registrando la incidencia.

Tareas y Subtareas

- **Tarea 1: Investigación y Selección del Protocolo SSO**
  - *Subtarea 1.1:* Investigar las opciones de integración SSO (OAuth2, SAML) y seleccionar la que se ajuste a los requerimientos de la universidad.
  - *Subtarea 1.2:* Documentar las especificaciones y configuraciones necesarias.
- **Tarea 2: Desarrollo del Módulo de Integración SSO en NestJS**
  - *Subtarea 2.1:* Configurar el módulo de autenticación en NestJS para soportar SSO (por ejemplo, utilizando Passport con estrategias SAML o OAuth2).
  - *Subtarea 2.2:* Implementar endpoints para redirección y callback del SSO.
  - *Subtarea 2.3:* Mapear la respuesta del proveedor SSO al modelo interno de usuario y generar un token JWT.
- **Tarea 3: Registro de Auditoría y Manejo de Excepciones**
  - *Subtarea 3.1:* Configurar logging para registrar cada autenticación vía SSO, incluyendo información relevante del proveedor.
  - *Subtarea 3.2:* Implementar manejo de excepciones para capturar errores en el flujo SSO y notificar a los administradores.
- **Tarea 4: Pruebas y Documentación**
  - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine con escenarios BDD (Given-When-Then) para el flujo SSO.
  - *Subtarea 4.2:* Documentar el proceso de integración SSO en la guía del usuario y la documentación técnica.
  - *Subtarea 4.3:* Integrar la funcionalidad SSO en el pipeline de CI/CD.

Should - Módulo de Auth (Seguridad y Control de Accesos)

RF-44: Registro de accesos y actividades para auditoría, que aporta trazabilidad y seguridad

HU-36: Registro de Accesos y Actividades para Auditoría

Historia de Usuario

Como **Administrador**, quiero **que el sistema registre de forma automática todos los accesos y actividades realizadas por los usuarios** para asegurar la trazabilidad de las acciones, detectar posibles incidentes de seguridad y facilitar auditorías internas y externas.

Criterios de Aceptación

- El sistema debe registrar cada inicio de sesión, cierre de sesión y acción relevante (creación, modificación, eliminación de datos, cambios de configuración, etc.).
- Cada registro debe incluir:
    - Identificación del usuario (ID, nombre, rol).
    - Fecha y hora exacta de la acción.
    - Tipo de acción realizada (login, logout, actualización, eliminación, etc.).
    - Dirección IP y, si es posible, información del dispositivo o navegador.
- Los registros deben ser inalterables y almacenarse de forma segura para garantizar su integridad.
- Se debe implementar una interfaz de consulta para que administradores autorizados puedan filtrar y visualizar los registros (por usuario, fecha, acción, etc.).
- La funcionalidad debe incluir la opción de exportar los registros en formato CSV para auditoría.
- La validación y registro de cada acción deben realizarse en menos de 2 segundos en condiciones normales.
- Toda acción de acceso y actividad debe quedar registrada en el historial de auditoría de la plataforma.

Para estructurar la solución, se desglosa en dos sub-historias:

SubHU-36.1: Registro Automático de Accesos y Actividades

Historia de Usuario

Como **Administrador**, quiero **que el sistema registre automáticamente todos los accesos y actividades críticas** para disponer de un historial completo y confiable que sirva como evidencia en auditorías y para el análisis de incidentes.

Criterios de Aceptación

- El sistema debe capturar y almacenar de manera automática cada inicio de sesión, cierre de sesión y acción relevante (por ejemplo, creación, edición o eliminación de recursos).

- Los registros deben incluir información detallada: usuario, rol, fecha, hora, tipo de acción, dirección IP y datos adicionales relevantes.
- Los registros deben almacenarse de forma segura (por ejemplo, en una base de datos de logs o mediante un servicio de logging centralizado).
- Se deben implementar validaciones que aseguren la integridad y consistencia de la información registrada.
- En caso de error en el registro, el sistema debe notificar al administrador y registrar la incidencia.
- La operación de registro no debe afectar significativamente el rendimiento del sistema (tiempo de registro < 2 segundos).

Tareas y Subtareas

- **Tarea 1: Diseño del Modelo de Datos para Auditoría**
  - *Subtarea 1.1:* Definir el esquema del log, incluyendo campos como ID de usuario, nombre, rol, fecha/hora, tipo de acción, IP y otros metadatos.
  - *Subtarea 1.2:* Validar el diseño del modelo con el equipo de seguridad y auditoría.
- **Tarea 2: Desarrollo del Módulo de Registro Automático**
  - *Subtarea 2.1:* Implementar un interceptor/middleware en NestJS que capture todas las solicitudes y acciones relevantes.
  - *Subtarea 2.2:* Desarrollar la lógica para almacenar la información capturada en la base de datos (MongoDB mediante Prisma) o enviarla a un sistema de logging centralizado (por ejemplo, Winston integrado con OpenTelemetry y Sentry).
  - *Subtarea 2.3:* Incluir validaciones de datos antes de guardar cada registro.
  - *Subtarea 2.4:* Realizar pruebas unitarias para asegurar la correcta captura y almacenamiento de registros.
- **Tarea 3: Manejo de Errores y Notificaciones**
  - *Subtarea 3.1:* Configurar manejo de excepciones para capturar errores en el proceso de registro.
  - *Subtarea 3.2:* Notificar a los administradores en caso de fallos críticos en el sistema de logging.
- **Tarea 4: Pruebas y Documentación**
  - *Subtarea 4.1:* Desarrollar pruebas de integración utilizando Jasmine con escenarios Given-When-Then.
  - *Subtarea 4.2:* Documentar el modelo de datos, la implementación del middleware y el flujo de registro en la guía técnica.
  - *Subtarea 4.3:* Integrar el módulo de auditoría en el pipeline de CI/CD (GitHub Actions, SonarQube, Pulumi).

SubHU-36.2: Consulta y Exportación del Historial de Auditoría

Historia de Usuario

Como **Administrador**, quiero **consultar y exportar el historial de accesos y actividades** para auditar el uso del sistema, detectar posibles incidentes y cumplir con las normativas de seguridad institucional.

Criterios de Aceptación

- La interfaz debe permitir visualizar una lista detallada de los registros de auditoría, con opciones de filtrado por usuario, fecha, tipo de acción, y dirección IP.
- Los administradores autorizados deben poder exportar los registros filtrados en formato CSV.
- La consulta debe responder en menos de 2 segundos en condiciones normales de carga.
- La interfaz debe ser responsiva y accesible tanto en dispositivos de escritorio como móviles.
- Todas las consultas y exportaciones deben quedar registradas en el historial de auditoría.
- Solo los usuarios con permisos adecuados pueden acceder a esta funcionalidad.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Consulta de Auditoría**
    - *Subtarea 1.1:* Crear wireframes y mockups de la pantalla de consulta de auditoría, incluyendo filtros y opciones de exportación.
    - *Subtarea 1.2:* Validar el diseño con administradores y el equipo de auditoría, ajustando según sea necesario.
- **Tarea 2: Desarrollo del Endpoint de Consulta de Auditoría**
    - *Subtarea 2.1:* Definir el DTO para la consulta que incluya filtros por usuario, fecha, tipo de acción, etc.
    - *Subtarea 2.2:* Implementar la lógica en el servicio para recuperar y formatear los registros de auditoría desde la base de datos.
    - *Subtarea 2.3:* Optimizar la consulta para asegurar un tiempo de respuesta inferior a 2 segundos.
- **Tarea 3: Desarrollo del Módulo de Exportación a CSV**
    - *Subtarea 3.1:* Implementar la lógica para generar un archivo CSV a partir de los datos filtrados.
    - *Subtarea 3.2:* Integrar la opción de exportación en la interfaz de consulta.
    - *Subtarea 3.3:* Validar el formato del CSV (encabezados, datos correctos).
- **Tarea 4: Registro de Auditoría y Manejo de Errores**
    - *Subtarea 4.1:* Configurar logging para registrar cada acción de consulta y exportación.
    - *Subtarea 4.2:* Implementar manejo de excepciones para capturar y notificar errores en el proceso de consulta/exportación.
- **Tarea 5: Pruebas y Documentación**
    - *Subtarea 5.1:* Desarrollar pruebas unitarias e integración utilizando Jasmine con escenarios BDD (Given-When-Then).
    - *Subtarea 5.2:* Documentar la funcionalidad de consulta y exportación en la guía del usuario y la documentación técnica.
    - *Subtarea 5.3:* Integrar la funcionalidad en el pipeline de CI/CD.

RF-45: Verificación de identidad en solicitudes críticas mediante autenticación de doble factor, fortaleciendo la seguridad

HU-37: Verificación de Identidad en Solicitudes Críticas mediante Doble Factor

Historia de Usuario

Como **Usuario con solicitudes críticas**, quiero **verificar mi identidad mediante autenticación de doble factor (2FA)** para asegurar que solo personal autorizado realice operaciones sensibles y fortalecer la seguridad del sistema.

Criterios de Aceptación

- La funcionalidad debe estar disponible para solicitudes consideradas críticas (definidas según reglas de negocio).
- Al activar una solicitud crítica, se debe solicitar una verificación 2FA adicional, mediante un código enviado a un canal configurado (SMS, correo electrónico o aplicación autenticadora).
- El usuario debe poder ingresar el código de 2FA en una interfaz segura.
- La verificación se considerará exitosa si el código ingresado es válido y se realiza dentro de un tiempo configurable (por ejemplo, 5 minutos).
- En caso de error (código inválido o vencido), el sistema mostrará un mensaje claro y permitirá reintentar la autenticación.
- Todas las acciones relacionadas (configuración de 2FA, intentos, verificaciones exitosas o fallidas) deben quedar registradas en el historial de auditoría.
- El tiempo de respuesta para la validación 2FA debe ser inferior a 2 segundos en condiciones normales.

Para abordar integralmente este requerimiento se desglosa en dos sub-historias:

SubHU-37.1: Configuración y Activación de Autenticación de Doble Factor

Historia de Usuario

Como **Usuario**, quiero **configurar y activar la autenticación de doble factor (2FA) en mi cuenta** para reforzar mi seguridad en el acceso a funciones críticas del sistema.

Criterios de Aceptación

- La interfaz de usuario debe permitir activar o desactivar la opción de 2FA desde el perfil.
- Debe ofrecer opciones para seleccionar el canal de verificación (SMS, correo electrónico o aplicación autenticadora).
- El proceso de configuración debe incluir la verificación inicial del canal seleccionado (por ejemplo, enviar un código de verificación).
- Una vez activado, la cuenta debe estar marcada como "2FA activado" y mostrar información relevante (última verificación, método seleccionado).
- El proceso de configuración y activación debe completarse en menos de 2 segundos y quedar registrado en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño de la Interfaz de Configuración 2FA**
  - *Subtarea 1.1:* Crear wireframes y mockups de la pantalla de configuración de 2FA en el perfil del usuario.
  - *Subtarea 1.2:* Incluir opciones de selección de canal (SMS, email, autenticador).
  - *Subtarea 1.3:* Validar el diseño con usuarios y stakeholders, ajustando según feedback.
- **Tarea 2: Desarrollo del Endpoint de Configuración de 2FA**
  - *Subtarea 2.1:* Definir el DTO para activar/desactivar 2FA, incluyendo la selección del canal y datos de verificación.
  - *Subtarea 2.2:* Implementar la lógica en el servicio de autenticación en NestJS para almacenar la configuración en la base de datos (MongoDB con Prisma).
  - *Subtarea 2.3:* Integrar con el servicio de mensajería (SMS/Email) o con una API para aplicaciones autenticadoras para enviar un código de verificación inicial.
  - *Subtarea 2.4:* Registrar la acción de activación en el historial de auditoría.
- **Tarea 3: Manejo de Excepciones y Validaciones**
  - *Subtarea 3.1:* Implementar validaciones que aseguren que el canal seleccionado esté disponible y funcione correctamente.
  - *Subtarea 3.2:* Desarrollar manejo de errores para notificar al usuario en caso de fallos en la configuración.
- **Tarea 4: Pruebas y Documentación**
  - *Subtarea 4.1:* Escribir pruebas unitarias e integración con Jasmine (escenarios Given-When-Then) para el flujo de configuración de 2FA.
  - *Subtarea 4.2:* Documentar el proceso de activación de 2FA en la guía del usuario y en la documentación técnica.
  - *Subtarea 4.3:* Integrar la funcionalidad en el pipeline de CI/CD.

SubHU-37.2: Verificación de Identidad en Solicitudes Críticas con 2FA

Historia de Usuario

Como **Usuario con solicitud crítica**, quiero **verificar mi identidad mediante un código 2FA al realizar una solicitud crítica** para asegurar que solo yo, como titular de la cuenta, pueda autorizar acciones sensibles y reforzar la seguridad.

Criterios de Aceptación

- Cuando un usuario realice una acción crítica, el sistema debe invocar automáticamente el flujo de verificación 2FA.
- Se enviará un código de verificación al canal configurado (SMS, email o autenticador).
- El usuario debe disponer de una interfaz segura para ingresar el código recibido.
- La verificación debe completarse en un tiempo configurable (por ejemplo, 5 minutos), y el sistema debe validar el código ingresado.

- Si la verificación es exitosa, la solicitud crítica se procesa; si falla, se permite reintentar o se cancela la acción, mostrando un mensaje de error claro.
- Todos los intentos, exitosos o fallidos, se registrarán en el historial de auditoría.

Tareas y Subtareas

- **Tarea 1: Diseño del Flujo de Verificación 2FA para Solicitudes Críticas**
    - *Subtarea 1.1:* Crear wireframes y mockups de la pantalla de ingreso del código de 2FA para acciones críticas.
    - *Subtarea 1.2:* Incluir mensajes de aviso sobre el tiempo límite y opciones de reintento.
- **Tarea 2: Desarrollo del Middleware de Verificación 2FA**
    - *Subtarea 2.1:* Definir el DTO para el envío del código de verificación y su validación.
    - *Subtarea 2.2:* Implementar en el servicio de autenticación la lógica que, al detectar una acción crítica, envíe el código a través del canal configurado.
    - *Subtarea 2.3:* Desarrollar el middleware o guard en NestJS que intercepte la solicitud crítica y requiera la validación 2FA antes de proceder.
    - *Subtarea 2.4:* Integrar la verificación del código, comparando el valor ingresado con el generado y registrando el tiempo de respuesta.
- **Tarea 3: Manejo de Excepciones y Notificaciones**
    - *Subtarea 3.1:* Implementar manejo de errores para notificar al usuario en caso de código incorrecto o expirado.
    - *Subtarea 3.2:* Registrar en el historial de auditoría todos los intentos de verificación (exitosos y fallidos).
- **Tarea 4: Pruebas y Documentación**
    - *Subtarea 4.1:* Desarrollar pruebas unitarias e integración con Jasmine utilizando escenarios Given-When-Then para validar el flujo 2FA en solicitudes críticas.
    - *Subtarea 4.2:* Documentar el proceso de verificación 2FA en la guía del usuario y en la documentación técnica.
    - *Subtarea 4.3:* Integrar la funcionalidad en el pipeline de CI/CD.