Trabajo práctico

Consigna:

Crear una aplicación del tipo servidor, usando como Framework principal, Nest JS (ver documentación), que satisfaga el siguiente requerimiento:

Escenario:

1. El cliente posee:
    a. 5 departamentos
    b. 10 parcelas
2. Aparte de eso, ya posee un microservicio para la autenticación de los usuario, por lo que la única necesidad es crear otro microservicio para su sistema de reservas.

Funcionalidades:

1. ***Sistema de Reservas (Departamentos)***:
    a. Los clientes deben poder ***registrar reservas*** para los departamentos.
    b. Las reservas inicialmente tendrán el estado ***PENDIENTE***, y la administración será responsable de ***APROBAR*** o ***DESAPROBAR*** las solicitudes.
    c. Se debe manejar la posibilidad de que haya ***más de dos reservas*** para la misma fecha, teniendo en cuenta los estados de aprobación. En caso que la reserva tenga estado ***APROBADA***, esas fechas ya no se podrían reservar, pero en estado pendiente, se podrían reservar, pero se aprueban solo una de ellas.
2. ***Gestión de Ingresos/Salidas (Parcelas)***:
    a. Cada cliente podrá registrar su ***ingreso*** a una parcela mediante un código único.
    b. El mismo código se utilizará para marcar la ***salida***.
    c. Si una parcela está ocupada, otro cliente no podrá registrar un ingreso hasta que se marque una salida (debe devolver un error 404).

Sockets:

1. La implementación de Sockets mejoraría la aplicación de la siguiente manera:
    a. Actualización en tiempo real del estado de las parcelas (***ingresos y salidas***) para los administradores.

Cuando hay ***dos usuarios conectados***, deben recibir actualizaciones en tiempo real sin necesidad de recargar la página, reflejando inmediatamente cualquier cambio en el estado de una parcela.

Requisitos:

1. **Validación de Variables de Entorno**: Utilizar el esquema de JOI para validar las variables de entorno.
2. **Modularización**: La aplicación debe estar correctamente modularizada, con separación clara de responsabilidades.
3. **Buenas Prácticas**:
    a. Implementar principios de ***responsabilidad única*** y ***clean code*** en toda la aplicación.
    b. Utilizar ***DTOs*** (Data Transfer Objects) junto con ***Class Validator*** para la validación de los datos entrantes en las solicitudes:
        i. Utilizar ***DTOs*** y ***Class Validator*** para validar los datos de entrada en las solicitudes de creación y modificación, asegurando que solo los datos correctos lleguen a la lógica de negocio.
        ii. Validar campos como ***fechas***, números, estados de reserva, etc.
    c. Implementar un ***paginador*** para las consultas a la base de datos (ej. al listar las reservas o parcelas).
    d. Incluir un manejo adecuado de ***errores***, asegurando que se devuelvan respuestas ***claras*** y ***apropiadas*** (por ejemplo, códigos de error HTTP y mensajes detallados):
        i. Asegurar que la aplicación devuelva códigos de error HTTP correctos (ej. 404 cuando una parcela está ocupada, 400 para datos inválidos, etc.).
        ii. Proporcionar mensajes claros al cliente cuando ocurran errores.
        iii. Incluir un sistema global de manejo de excepciones (por ejemplo, un filter en NestJS).
4. **Base de Datos**: Utilizar ***TypeOrm*** o ***Prisma*** (ver documentación) para la administración de la base de datos.
5. **Opcional**: Integrar ***Socket.IO*** o ***WebSocket*** para la comunicación en tiempo real (se valorará el esfuerzo adicional).