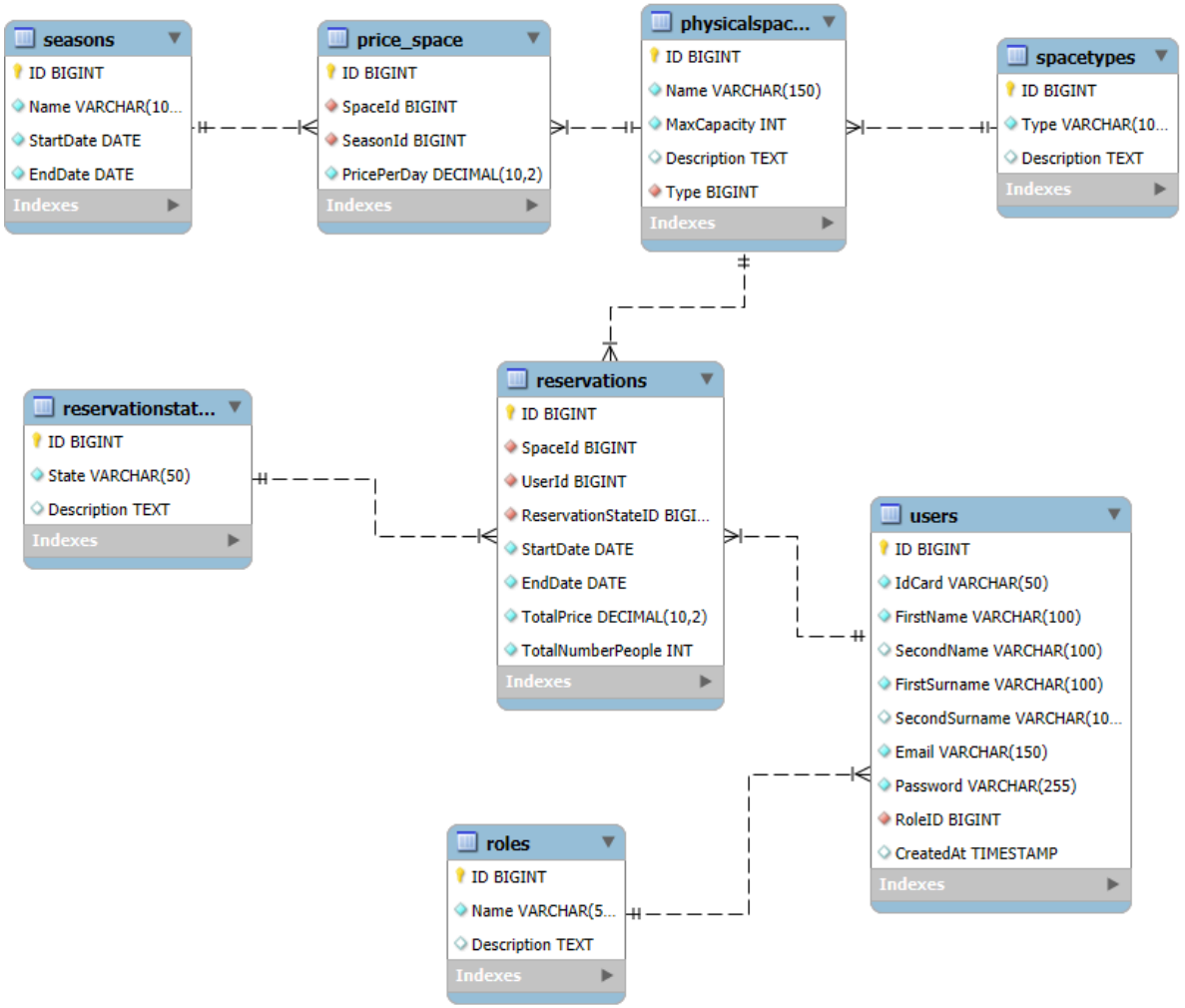
Realizado por Jesús Gabriel Parra Dugarte (jgparra@unicauca.edu.co)

el código de la solución se encuentra en el siguiente repositorio git:
https://github.com/ParraJesus/prueba-tecnica.git

Punto 1.

Entregable 1: Realice un modelo que incluya tablas y relaciones de la base de datos que darán solución a la aplicación planteada.



Entregable 2: Describa la arquitectura con la cual implementaría esta solución.

La solución para el centro comercial “El Futuro” permite gestionar los espacios físicos disponibles para alquiler, los usuarios de la plataforma, las reservas y las notificaciones. La arquitectura propuesta garantiza escalabilidad, modularidad y alta disponibilidad, permitiendo que el sistema crezca según la demanda y pueda integrarse con servicios externos.

Arquitectura general. Se propone una arquitectura basada en microservicios, donde cada componente del backend tiene responsabilidades bien definidas y se comunica a través de APIs REST o colas de mensajería. La solución se organiza en tres capas principales:

1. Capa de presentación (Frontend)
2. Capa de negocio (Backend: microservicios)
3. Capa de persistencia (Bases de datos)

El entorno de despliegue se realiza con contenedores Docker, orquestados mediante Docker Compose. En un entorno cloud, se pueden usar servicios como AWS Elastic Container Service (ECS), Elastic Compute Cloud (EC2) o AWS Fargate para una implementación más serverless.

Frontend.
Aplicación web: desarrollada en React, destinada a usuarios y administradores para interactuar con la plataforma desde un navegador.

App móvil: desarrollada en React Native, permite a los usuarios consultar disponibilidad de espacios, realizar reservas y recibir notificaciones desde cualquier dispositivo.

Funcionalidades:
- Búsqueda de espacios por fecha, tipo y capacidad.
- Gestión de reservas y cancelaciones.
- Visualización de reportes para administradores.
- Recepción de notificaciones de manera asíncrona.

Backend: Microservicios (Spring Boot) Cada microservicio se despliega de manera independiente y puede escalar según la demanda.

Gestión de usuarios:
- Autenticación y autorización de usuarios y administradores.
- Cifrado seguro de contraseñas.
- Creación, edición y eliminación de usuarios y roles.

Gestión de espacios:
- CRUD de espacios físicos y sus tipos.
- Configuración de precios por temporada.
- Validación de capacidad y disponibilidad.

Gestión de reservas:
- Creación y actualización de reservas.
- Cálculo automático de precio según temporada.
- Cancelaciones con penalización del 10% del valor total.
- Este microservicio puede escalar horizontalmente según el número de reservas concurrentes.

Notificaciones:
- Envía correos electrónicos, mensajes SMS o push notifications.
- Conectado a una cola de mensajería (RabbitMQ) para procesamiento asíncrono.
- Persistencia de mensajes en MongoDB, permitiendo historial y trazabilidad

Base de datos:

Componente	Tipo de base de datos	Descripción
Usuarios, Espacios, Reservas, Tipos, Temporadas, Estados	MySQL	Base de datos relacional principal para garantizar integridad referencial y consistencia de datos.
Notificaciones	MongoDB	Base de datos NoSQL para almacenar mensajes, notificaciones y logs, facilitando consultas rápidas y escalabilidad horizontal.

Flujo de datos

- El usuario inicia sesión a través de la web o app móvil > Backend de Gestión de usuarios valida credenciales.
- El usuario busca disponibilidad de un espacio > Backend de Gestión de espacios consulta la base de datos MySQL para devolver resultados.
- El usuario realiza la reserva > Backend de Gestión de reservas registra la reserva, calcula el precio según temporada y actualiza el estado.
- Si la reserva se cancela > Backend de reservas actualiza el estado y calcula la penalización.
- Todas las acciones que requieren comunicación asincrónica > Backend de Notificaciones envía mensajes a través de RabbitMQ, que luego se almacenan en MongoDB para historial y procesamiento.
- Los reportes de administración se generan a partir de consultas agregadas a la base de datos relacional o en el backend.

5. Seguridad y control de acceso

- Contraseñas cifradas con algoritmos seguros como bcrypt.
- Roles de usuario y administrador para control de acceso a funciones críticas.
- Validación de datos en backend: fechas, capacidad de espacios, disponibilidad.
- Comunicación segura mediante HTTPS entre frontend y backend.

6. Escalabilidad y despliegue

Microservicios independientes: cada componente puede escalar horizontalmente según demanda (por ejemplo, el microservicio de reservas).

Contenedores Docker: facilitan despliegues consistentes, replicables y portables.

Orquestación: Docker Compose para desarrollo local; en producción, ECS o Fargate en AWS permiten escalado automático y gestión serverless.

Bases de datos escalables:
MySQL para datos críticos relacionales, con backups y réplicas. MongoDB para notificaciones, escalable horizontalmente para manejar altos volúmenes de mensajes.