Alumno—
EDUARDO VALDEZ RUBIO

ID—
00000148297

Asignación—
Proyecto 3

Fecha—
10 de diciembre de 2025

Materia—
Bases de datos avanzadas

Profesor—
Martin Guadalupe Bernal Lugo

Proyecto: Ticket Hub

La industria del entretenimiento y la venta de boletos en línea opera bajo condiciones extremas de tráfico web, conocidas como "picos de alta concurrencia".

En un sistema tradicional basado en Bases de Datos Relacionales (SQL), la información de un evento suele estar fragmentada en múltiples tablas normalizadas.

Para mostrar la página de un solo concierto a un usuario, el sistema debe ejecutar operaciones de unión (JOINs) costosas entre todas estas tablas cuando miles de usuarios intentan acceder simultáneamente a la información de un evento popular, la base de datos relacional se satura intentando reconstruir la información mediante JOINs para cada usuario.

La rigidez del esquema SQL impide adaptar fácilmente la estructura de datos para diferentes tipos de eventos (ej. un concierto numerado vs. un festival de varios días con pases generales sin alterar la estructura global de la base de datos.

La problemática impacta directamente a dos grupos principales:

1. Usuarios Finales: Son quienes intentan comprar boletos. Se ven afectados por tiempos de carga lentos y caídas de sistema justo en el momento de la compra, o errores de inconsistencia donde un boleto parece disponible pero ya no lo está.
2. Organizadores del Evento: Sufren pérdidas económicas directas si el sistema cae. Además, enfrentan limitaciones operativas al no poder configurar estructuras de precios flexibles o promociones dinámicas debido a la rigidez de la base de datos.

Resolver este problema es crítico para la viabilidad del negocio. En la era digital, la disponibilidad y la velocidad no son lujos, son requisitos funcionales. Garantizar que el sistema pueda soportar picos de tráfico masivos asegura la rentabilidad del evento y la confianza del consumidor. Además, permitir una gestión de datos flexible es vital para adaptarse a las nuevas modalidades de entretenimiento híbrido y dinámico.

La migración de la lógica de negocio a una arquitectura basada en documentos con MongoDB no es solo una elección de mejora en fiabilidad, sino una respuesta técnica

directa a los requisitos no funcionales de alto rendimiento en lectura y flexibilidad de modelado que exige un sistema moderno de venta de boletos como TicketHub.

Para TicketHub, decidimos usar MongoDB en lugar de una base de datos tradicional (como MySQL) porque se adapta mucho mejor a la naturaleza cambiante y masiva de un sistema de eventos.

- Flexibilidad: En el mundo de los eventos, no todo es igual. Un concierto puede tener más o menos asientos, mientras que un festival puede ser de entrada general gratis por días. Con MongoDB, no estamos atados a una estructura rígida de tablas. Podemos guardar información diferente para cada tipo de evento sin tener que rediseñar toda la base de datos cada vez que algo cambia.
- Escalabilidad: Imagina que sale a la venta el concierto del año y miles de personas entran al mismo tiempo. Las bases de datos tradicionales a veces "sufren" con tantos usuarios simultáneos. MongoDB está diseñado para crecer horizontalmente (es decir, añadir más servidores baratos para repartir la carga) de manera mucho más sencilla. Esto es vital para que TicketHub no se caiga justo cuando más ventas estamos teniendo.
- Manejo de documentos embebidos: En lugar de tener la información regada en muchas tablas (una para el evento, otra para el lugar, otra para los precios) y tener que usarlas cada vez que alguien consulta un evento (lo que es lento), en MongoDB guardamos todo junto en un solo "documento". Por ejemplo, la información del estadio y los precios están *dentro* del documento del concierto. Esto hace que leer la información sea rapido, porque con una sola búsqueda traemos todo lo necesario para mostrar la página.
- Facilidad para representar relaciones no estructuradas: A veces los datos no siguen un patrón perfecto. Por ejemplo, un usuario puede tener un historial de compras simple, y otro puede usar códigos de descuento. MongoDB nos permite guardar toda esta información variada y compleja de forma natural, tal como la usamos en la aplicación, sin tener que forzarla en filas y columnas que no siempre tienen sentido.

Diseño de la Base de Datos (MongoDB)

Para el proyecto TicketHub, hemos diseñado un esquema NoSQL utilizando MongoDB, que se estructura en tres colecciones principales. Este diseño aprovecha la flexibilidad de los documentos JSON para almacenar información compleja de manera anidada, optimizando las consultas de lectura.

A. Colección usuarios

Esta colección almacena la información de todos los usuarios registrados en la plataforma, tanto clientes como personal administrativo.

- Nombre de la colección: users
- Documento Padre (Raíz): Representa a un usuario individual.
- Atributos:
    - _id: ObjectId (Identificador único generado por MongoDB).
    - username: String (Nombre de usuario único para inicio de sesión).
    - password: String (Contraseña encriptada).
    - type: String (Rol del usuario: 'client' o 'staff').
    - eventsBooked: Array de Numbers (Lista de IDs de eventos (eventId) que el usuario ha comprado. Esto permite una búsqueda rápida de los eventos activos del usuario sin tener que consultar la colección de tickets).
    - history: Array de Documentos Embebidos (Historial de actividad del usuario).
        - Documento Embebido (history):
            - action: String (Descripción de la acción, ej. "Boletos comprados para Concierto X").
            - time: Date (Fecha y hora de la acción).

B. Colección eventos

Esta es la colección central del sistema. Almacena toda la información relacionada con los conciertos y festivales. Utiliza documentos embebidos para evitar la necesidad de múltiples tablas (como Recintos o Precios) que serían necesarias en SQL.

- Nombre de la colección: events
- Documento Padre (Raíz): Representa un evento específico.
- Atributos:
    - _id: ObjectId (Identificador interno de MongoDB).
    - eventId: Number (Identificador numérico legible para URLs y referencias rápidas).
    - eventName: String (Nombre del evento).
    - summary: String (Descripción corta).
    - fullDesc: String (Descripción detallada, puede contener HTML).
    - address: String (Dirección del recinto; simplificado como string para este MVP, pero conceptualmente representa la información del lugar).
    - startDate: Date (Fecha y hora de inicio).
    - endDate: Date (Fecha y hora de finalización).
    - capacity: Number (Aforo total del evento).
    - currentBookings: Number (Contador de boletos vendidos actualmente).

- price: Number (Precio general del boleto).
- promoCode: String (Código promocional opcional).
- discount: Number (Porcentaje de descuento si aplica).

C. Colección tickets (Boletos)

Esta colección registra cada transacción de compra individual. Actúa como la entidad que vincula a un Usuario con un Evento, permitiendo el control de acceso y el historial detallado.

- Nombre de la colección: tickets
- Documento Padre (Raíz): Representa un boleto único emitido.
- Atributos:
  - _id: ObjectId (Identificador único del boleto).
  - eventId: Number (Referencia al eventId de la colección events).
  - purchaseDate: Date (Fecha y hora de la compra).
  - status: String (Estado del boleto: 'VALID', 'USED', 'CANCELLED').
  - user: ObjectId (Referencia al _id de la colección users).
  - event: ObjectId (Referencia al _id de la colección events).

Relación entre Colecciones

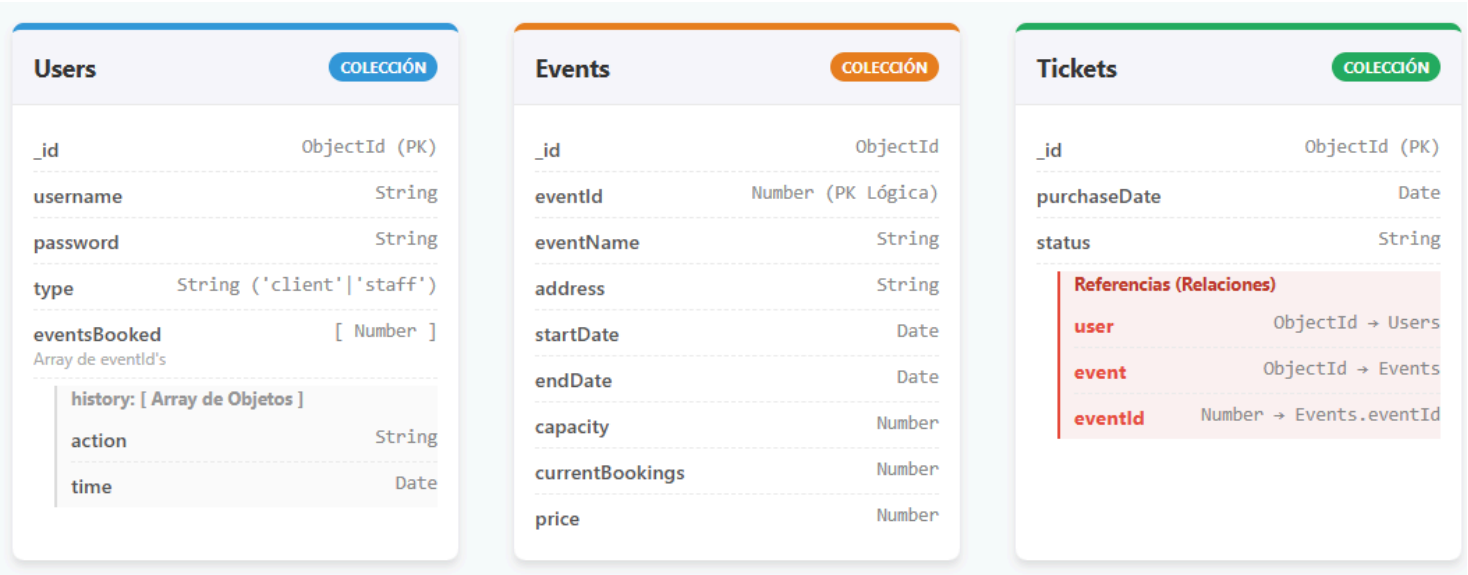
Aunque MongoDB es no relacional, existen referencias lógicas entre los documentos:

1. Usuarios - Eventos: La relación es de "muchos a muchos". Un usuario puede asistir a muchos eventos, y un evento tiene muchos usuarios.
   - *Implementación:* Se maneja de forma híbrida.
     - En users, el array eventsBooked guarda una lista rápida de los IDs de eventos.
     - La colección tickets actúa como una "tabla de unión" detallada, guardando la relación específica de una compra.
2. Tickets - Usuarios/Eventos:
   - El documento Ticket contiene referencias directas (ObjectId) tanto al documento del User como al del Event.

Este diseño cumple con los requisitos de la materia al demostrar el uso de:

- Colecciones: users, events, tickets.
- Documentos Embebidos: El historial dentro de users.
- Arrays: La lista eventsBooked en users.

- Referencias: Los campos ObjectId en tickets para conectar datos.



Experiencia personal en el proyecto

Trabajar con MongoDB fue un cambio de mentalidad interesante. Al venir de un fondo de bases de datos SQL, al principio costaba un poco no pensar en tablas y FKs. Sin embargo, aprendí a valorar la libertad que te da el modelo de documentos. Fue muy útil ver cómo podíamos guardar toda la información de un evento en un solo objeto JSON, lo que hizo que las consultas en el código fueran mucho más directas y rápidas

La principal dificultad fue la integración inicial y la configuración del entorno. Tuve varios problemas técnicos al conectar la aplicación Node.js con MongoDB Atlas, especialmente con las versiones de las librerías y la gestión de sesiones de usuario (express-session). También fue un reto manejar correctamente la lógica asíncrona async/await para asegurar que, al cancelar un boleto, se actualizarán correctamente tres cosas a la vez: el contador del evento, el historial del usuario y el estado del boleto, sin dejar datos inconsistentes.

Si tuviera más tiempo me hubiera gustado implementar 3 nuevas funciones:

1.- Que se generará un ticket con QR, a su vez que ese QR también se guarde en la colección de tickets , en una versión que el ticket se pueda guardar como imagen común.

2.-Escaneo de QRs: Desarrollar una pequeña app móvil o una vista de administrador que use la cámara del celular para escanear los códigos QR de los boletos y validar su entrada en tiempo real.

3.-Mapa de Asientos Interactivo: En lugar de solo vender entrada general, me gustaría usar un canvas HTML5 para que el usuario pueda elegir su asiento específico en un mapa visual del recinto y que a su vez el staff a la hora de crear un evento pueda proporcionar esta información espacial del lugar.