Database Management Final Project

Title of the Project: EventHive

Manager: Ridhima Reddy

Team Members: Curt Leonard, Heeya Jolly, Jason Maurer, Thomas Nguyen

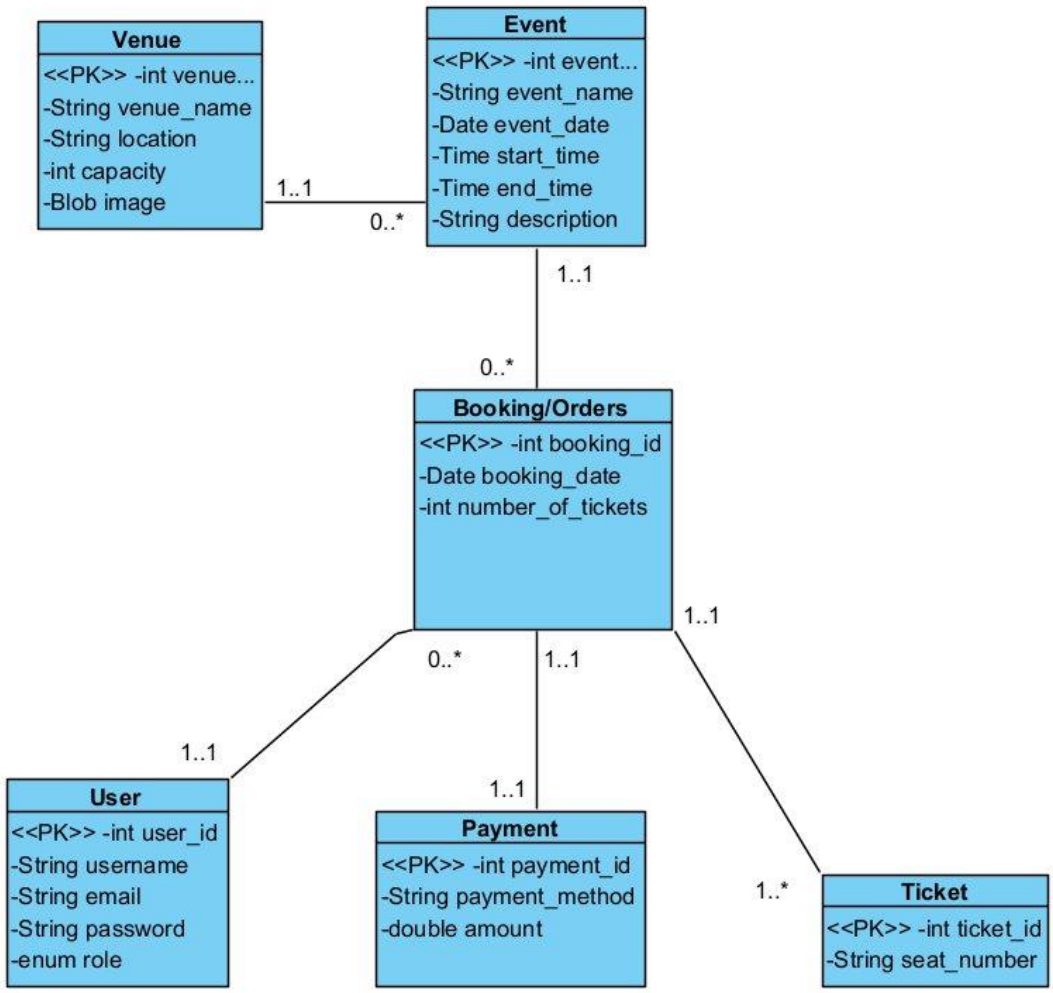
Project Overview:
The Event Management System is designed to simplify the process of organizing and managing events. Its goal is to provide a web-based platform where users can create, schedule, and manage events, invite participants, and track RSVPs. The system helps both event organizers and participants stay organized, ensuring smooth event planning and execution. It reduces the need for manual communication and tracking, making event management more efficient.

Core Functionalities

1. **User Management**
    a. Users can register, log in, and manage their profiles.
    b. Roles include:
        i. **Event Organizers(admin):** Create and manage events.
        ii. **Regular Users:** RSVP and participate in events.
2. **Event Creation and Management**
    a. Organizers can:
        i. Add, edit, or delete events.
        ii. Define event details: date, time, location, description, and ticket price.
3. **RSVP Tracking**
    a. Participants can RSVP for events.
    b. Organizers can monitor attendance and RSVPs.
4. **Notifications**
    a. Automatic email notifications are sent to participants for:
        i. Event reminders.
        ii. Event updates (e.g., cancellations or time changes).
5. **Post-Event Feedback**
    a. Participants can provide feedback after events.
    b. Organizers can view feedback to improve future event experiences.

Design and Implementation Details

UML Class Diagram



The UML Class Diagram shows the main components and their relationships in the Event Management System, designed to streamline event planning and execution.

Key Classes and Attributes:

1. Venue:
    a. Represents the location where events are held.
    b. Attributes:
        i. venue_id: Unique identifier for the venue.

ii. venue_name: Name of the venue.
iii. location: Venue's physical location.
iv. capacity: Maximum number of attendees.
v. image: A visual representation of the venue (stored as a blob).

2. Event:
    a. Represents individual events organized at a venue.
    b. Attributes:
        i. event_id: Unique identifier for the event.
        ii. event_name: Name of the event.
        iii. event_date: Date of the event.
        iv. start_time: Event start time.
        v. end_time: Event end time.
        vi. description: A brief description of the event.
3. Booking/Orders:
    a. Represents a booking or ticket order for an event.
    b. Attributes:
        i. booking_id: Unique identifier for the booking.
        ii. booking_date: Date when the booking was made.
        iii. number_of_tickets: Number of tickets in the booking.
4. User:
    a. Represents individuals who use the system (e.g., event organizers or attendees).
    b. Attributes:
        i. user_id: Unique identifier for the user.
        ii. username: The user's name.
        iii. email: The user's email address.
        iv. password: User's encrypted password.
        v. role: User's role in the system (e.g., attendee, organizer).
5. Payment:
    a. Represents payment details for a booking.
    b. Attributes:
        i. payment_id: Unique identifier for the payment.
        ii. payment_method: Method of payment (e.g., credit card, PayPal).
        iii. amount: Total amount paid.
6. Ticket:
    a. Represents individual tickets within a booking.
    b. Attributes:
        i. ticket_id: Unique identifier for the ticket.

ii. seat_number: Assigned seat number for the ticket.

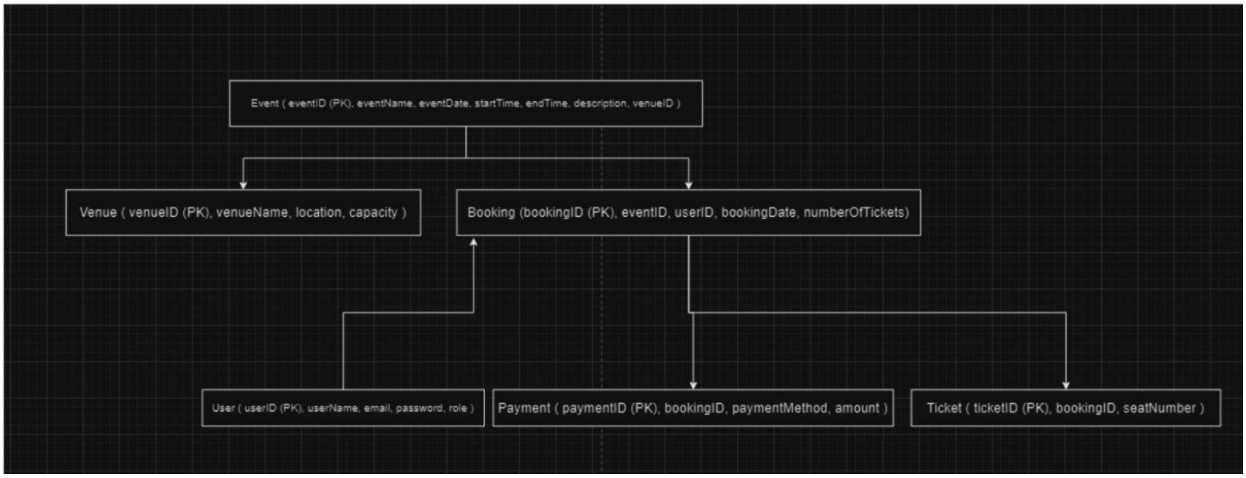
Relationships:

- Venue ↔ Event:
  - A venue can host multiple events (1..*).
  - Each event is linked to a single venue (1..1).
- Event ↔ Booking/Orders:
  - An event can have multiple bookings (0..*).
  - Each booking is linked to a specific event (1..1).
- Booking/Orders ↔ User:
  - A user can make multiple bookings (0..*).
  - Each booking is associated with one user (1..1).
- Booking/Orders ↔ Payment:
  - A booking has exactly one payment record (1..1).
  - Each payment is linked to a specific booking (1..1).
- Booking/Orders ↔ Ticket:
  - A booking can generate multiple tickets (1..*).
  - Each ticket belongs to a single booking (1..1).

Summary:

This diagram provides a clear and structured overview of how venues, events, bookings, users, payments, and tickets are interrelated. It reflects the system's ability to manage venues, schedule events, process user bookings, handle payments, and issue tickets efficiently.

Schema 1: Relational Model

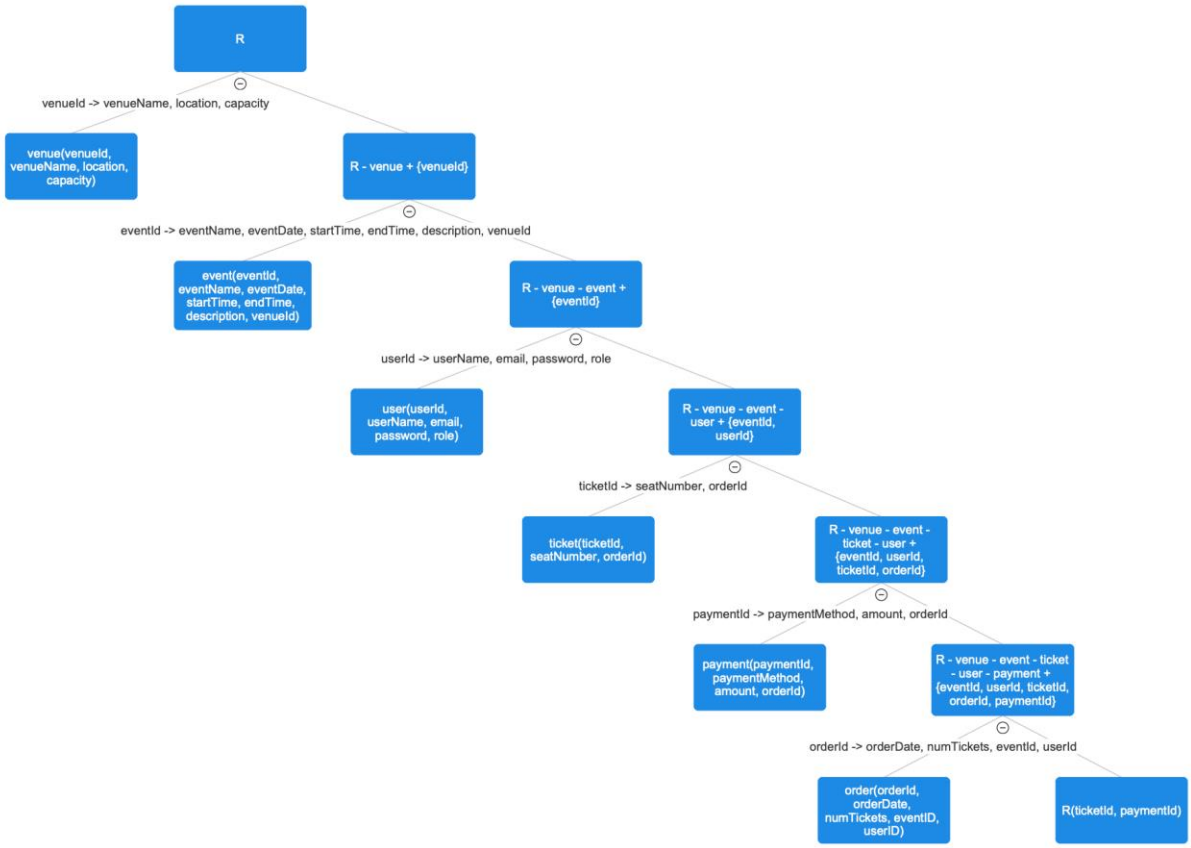


Relational Model Translation

The UML diagram is translated into relational tables:

- Event(eventId, eventName, eventDate, startTime, endTime, description, price, venueId)
- User(userId, userName, email, password, role)
- Venue(venueId, venueName, capacity, location)
- Order(orderId, userId, eventId, numberOfTickets, orderDate)
- Ticket(ticketId, orderId, seatNumber)
- Payment(paymentId, orderId, paymentType, amount)

Schema 2: BCNF



BCNF Overview:

BCNF is designed to reduce redundancy and ensure data integrity. It is organized into 7 tables. The Venue table stores details about venues, including a unique venueId, name, location, and capacity. The Event table records event-specific information like eventId, name, date, time, description, and the associated venueId. Users are managed in the User table, which includes userId, username, email, password, and role (e.g., organizer or participant). The Order table tracks bookings with details like orderId, date, number of tickets, the associated eventId, and userId. Payments are handled in the Payment table, which includes paymentId, payment method, amount, and the corresponding orderId. Tickets are stored in the Ticket table with ticketId, seat number, and the related orderId. These tables are connected through foreign keys, ensuring relationships such as a venue hosting many events, a user making multiple orders, and each order having tickets and payments. This structure eliminates anomalies, making the system efficient and reliable.

Schema 3: 3NF Synthesis

Create minimal cover

EventId -> eventName, eventId -> eventDate, eventId -> startTime, eventId -> endTime, eventId ->description, eventId -> venueId
VenueId -> venueName, venueId -> location, venueId -> capacity
OrderId -> orderDate, orderId -> numberOfTickets, orderId -> eventId, orderId -> userId
UserId -> userName, userID -> email, userId -> password, userId -> role
PaymentId -> paymentMethod, paymentId -> amount, paymentId -> orderId
TicketId -> seatNumber, ticketId -> orderId

No redundant/extraneous attributes

3NF Synthesis – Create Tables for Each FD, and add Key Table

- Event(eventId, eventName, eventDate, startTime, endTime, description, price, venueId)
- Venue(venueId, venueName, location, capacity)
- Order(orderId, orderDate, numberOfTickets, eventId, userId)
- User(userId, userName, email, password, role)
- Payment(paymentId, paymentMethod, amount, bookingId)
- Ticket(orderId, ticketId, seatNumber)
- Key(paymentId, ticketId)

No subsets of other tables to remove, so this is the final schema

Software Architecture and Components

The software architecture and components of EventHive were carefully designed to ensure a seamless and secure event management experience. For the frontend, we utilized React.js and Next.js frameworks, along with HTML, CSS and JavaScript to develop the user interface. The design is responsive, ensuring usability across multiple devices. The backend was implemented in Java, using Springboot API framework and Hibernate ORM to handle server-side logic and processing, while MySQL serves as the relational database for storing and managing data.

We prioritized security by implementing robust mechanisms to protect user data:

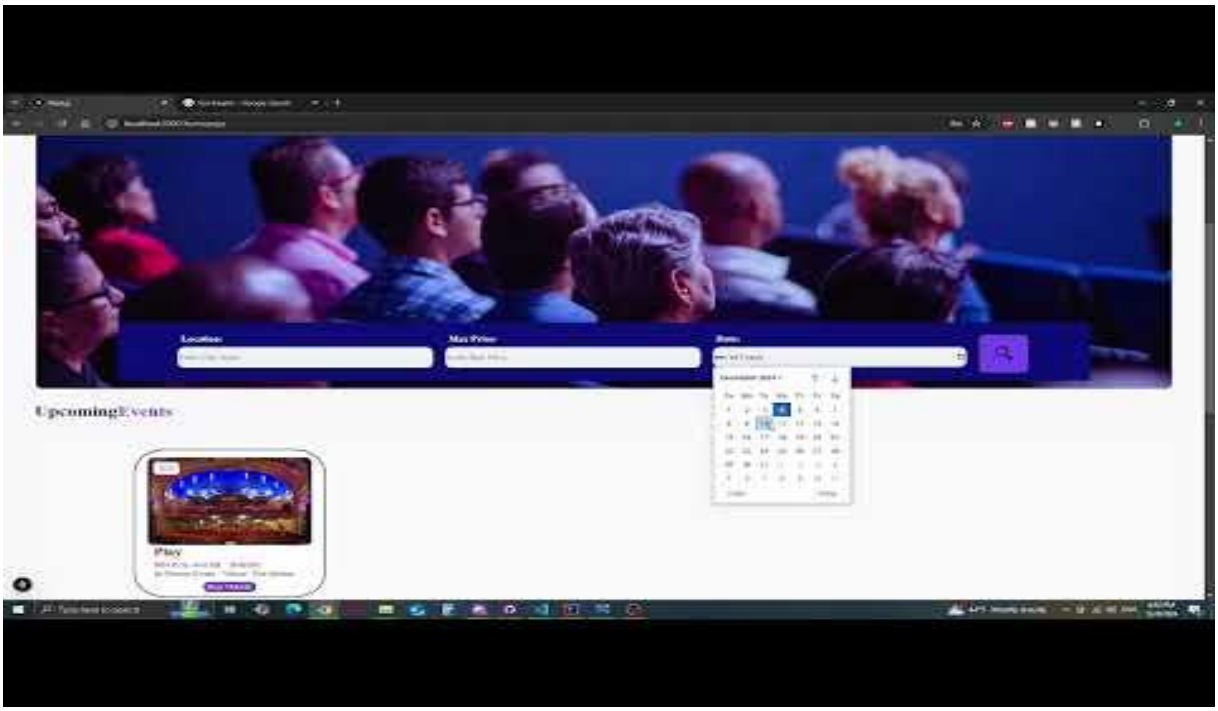
- Password Handling:

    - We integrated the BCrypt Cryptography Library to hash passwords with a random salt using the Blowfish Cypher Algorithm.
    - On user login, passwords are verified securely using the hashed values.
    - Value: If the database is compromised, the hacker would need to compute a unique rainbow table for each database entry, significantly increasing the difficulty of decrypting passwords.
- SQL Injection Prevention:
    - By leveraging our ORM Hibernate, we query the database exclusively using parameterized SQL queries.
    - Value: This prevents raw SQL input, eliminating vulnerabilities that could be exploited through malicious user input.

To connect the frontend and database, we designed a robust API layer to facilitate smooth communication between the two. For user interface design, we utilized Figma to create and link the user interface, providing an intuitive and visually appealing experience for users.

This comprehensive architecture, with a strong focus on security and usability, guarantees a reliable, user-friendly, and secure platform for event management.

Link to Demo: https://www.youtube.com/watch?v=FEtOobGoKks


Location
Max Price
Date
UpcomingEvents
Play